6. Identity testing and probabilistically checkable proofs

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

Lecture schedule

Tue 15 Jan:	1. Polynomials and integers
Tue 22 Jan:	2. The fast Fourier transform and fast multiplication
Tue 29 Jan:	3. Quotient and remainder
Tue 5 Feb:	4. Batch evaluation and interpolation
Tue 12 Feb:	5. Extended Euclidean algorithm and interpolation from erroneous data
Tue 19 Feb:	*Exam week — no lecture*
Tue 27 Feb:	6. Identity testing and probabilistically checkable proofs
Tue 5 Mar:	*Break — no lecture*
Tue 12 Mar:	7. Finite fields
Tue 19 Mar:	8. Factoring polynomials over finite fields
Tue 26 Mar:	9. Factoring integers

CS-E4500 Advanced Course in Algorithms (5 ECTS, III–IV, Spring 2019)

2019	KALENTERI	2019

	Tammikuu	Helmikuu	Maaliskuu	Huhtikuu	Toukokuu	Kesäkuu
1	Ti Uudenvuodenpäivä	Pe	Pe	Ma Vk 14 T9	Ke Vappu	La
2	Ke	La	La	Ti	To	Su
3	To	Su D3	Su	Ke	Pe	Ma Vk 23
4	Pe	Ma Vk 06 T3	Ma Vk 10	To	La	Ti
5	La	Ti L4	Ti Laskiainen	Pe	Su	Ke
6	Su Loppiainen	Ke	Ke Break	La	Ma Vk 19	To
7	Ma Vk 02	To Q4	To	Su	Ti	Pe
8	Ti	Pe	Pe	Ma Vk 15	Ke	La
9	Ke	La	La	Ti	To	Su Helluntaipäivä
10	To	Su D4	Su D6	Ke	Pe	Ma Vk 24
11	Pe	Ma Vk 07 T4	Ma Vk 11 T6	To	La	Ti
12	La	Ti L5	Ti L7	Pe	Su Äitienpäivä	Ke
13	Su	Ke	Ke	La	Ma Vk 20	To
14	Ma Vk 03	To Q5	To Q7	Su Palmusunnuntai	Ti	Pe
15	Ti L1	Pe	Pe	Ma Vk 16	Ke	La
16	Ke	La	La	Ti	To	Su
17	To Q1	Su	Su D7	Ke	Pe	Ma Vk 25
18	Pe	Ma Vk 08	Ma Vk 12 T7	To	La	Ti
19	La	Ti Exam	Ti L8	Pe Pitkäperjantai	Su Kaatuneiden muistopäivä	Ke
20	Su D1	Ke	Ke Kevätpäiväntasaus	La	Ma Vk 21	To
21	Ma Vk 04 T1	To week	To Q8	Su Pääsiäispäivä	Ti	Pe Kesäpäivänseisaus
22	Ti L2	Pe	Pe	Ma 2. pääsiäispäivä	Ke	La Juhannus
23	Ke	La	La	Ti	To	Su
24	To Q2	Su D5	Su D8	Ke	Pe	Ma Vk 26
25	Pe	Ma Vk 09 T5	Ma Vk 13 T8	To	La	Ti
26	La	Ti L6	Ti L9	Pe	Su	Ke
27	Su D2	Ke	Ke	La	Ma Vk 22	To
28	Ma Vk 05 T2	To Q6	To Q9	Su	Ti	Pe
29	Ti L3		Pe	Ma Vk 18	Ke	La
30	Ke		La	Ti	To Helatorstai	Su
31	To Q3		Su Kesäaika alkaa D9		Pe	

L = Lecture; hall T5, Tue 12–14

Q = Q & A session; hall T5, Thu 12–14

D = Problem set deadline; Sun 20:00

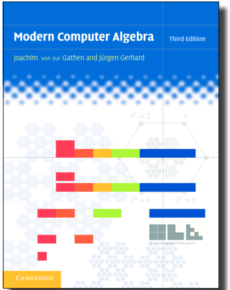
T = Tutorial (model solutions); hall T6, Mon 16–18

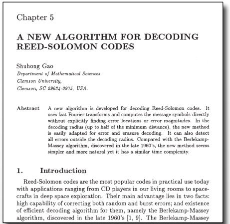
Recap of last week

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm for polynomials by **divide and conquer**
  - The two polynomial operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)
- Coping with **errors in data** using **error-correcting codes**
- A family of error-correcting codes (**Reed–Solomon codes**) based on evaluation–interpolation duality for univariate polynomials
  - Key observation: low-degree polynomials have few roots (exercise)
  - Fast **encoding** and **decoding** of Reed–Solomon codes via the fast univariate polynomial toolkit and **Gao's** (2003) **decoder**

Have: Near-linear-time toolbox for univariate polynomials

- Multiplication
- Division (quotient and remainder)
- Batch evaluation
- Interpolation
- Extended Euclidean algorithm (gcd)
- Interpolation from partly erroneous data





Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-0975, USA.

Abstract A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

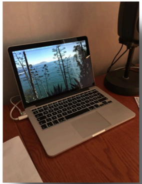
Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Motivation for this week

- Last week we encountered **uncertainty** in computation
- We saw how to cope with uncertainty in the form of **errors in data** by using **error-correcting codes**
- This week we look at (fine-grained) **proof systems** and **errors in computation** …
- Our motivation is to be able to **delegate computation** …

Delegating computation

Client



modest resources
reliable

Problem instance

Solution

Service-provider

massively SIMD-parallel resources
error-prone

- How to verify that the solution is correct ?

- How to design an algorithm to tolerate (a small number of) errors *during computation* ?

- How to convince the client or a third party that the solution is correct ?

Key content for Lecture 6

- We look at yet further applications of the evaluation–interpolation duality and randomization in algorithm design
- Randomized **identity testing** for polynomials and matrices (exercise)
- **Delegating computation** and **proof systems**
- **Completeness** and **soundness** of a proof system,
  cost of **preparing** a proof, cost of **verifying** a proof
- Williams's (2016) [30] probabilistic proof system for #CNFSAT
- Coping with **errors in computation** using error-correcting codes with multiplicative structure (Reed–Solomon codes revisited)
- Proof systems that tolerate errors during proof preparation (Björklund & K. 2016) [3]
- An extension of Shamir's secret sharing to delegating a computation to multiple counterparties (delegating matrix multiplication, exercise)

Proof systems

- Let I be a **claim**
  (an instance of a computational problem with a yes/no (true/false) solution)
- Let us assume that I is decidable, that is, there exists an algorithm D that given I as input outputs whether I is true
- Deciding whether I is true can often be assisted by supplying a **proof** Π for I
- A **proof system** consists of a verification algorithm (the **verifier**) V that takes as input I together with a putative proof $\tilde{\Pi}$ and either accepts or rejects $\tilde{\Pi}$ as a proof for I

Completeness and soundness

- A proof system with verifier V is
  - **complete** if for every true I there exists a proof Π such that V accepts on input I and Π
  - **sound** if for every false I and every putative proof $\tilde{\Pi}$ it holds that V rejects on input I and $\tilde{\Pi}$

Probabilistic soundness

- Let us relax the notion of soundness somewhat by allowing the verifier V to make random choices during its execution
- A proof system with a randomized verifier V is **probabilistically sound** if for every false I and every putative proof $\tilde{\Pi}$ it holds that V rejects with high probability on input I and $\tilde{\Pi}$
- By “high probability” we mean with probability $1 - o(1)$ as a function of the size of I, where probability is over the random choices made by V

Efficiency (verifier)

- In addition to completeness and soundness, in general we want a proof system also to be *efficient*
- That is, V on input I and $\tilde{\Pi}$ should consume less computational resources than it takes to decide I (using the best known algorithm for deciding I)

Efficiency (prover)

- Besides verifier efficiency, a yet further aspect to a proof system are the computational resources to **prepare** a proof
- Let P be an algorithm (the **prover**) that given a claim I as input outputs whether I is true, and if I is true, also outputs a proof Π such that V accepts on input I and Π
- We would like P to be efficient in the sense that P should not consume substantially more computational resources than it takes to decide I (using the best known algorithm for deciding I)

(Some of) recent work on fine-grained proof systems

- Goldwasser, Kalai, Rothblum [12]
- Walfish and Blumberg [29]
- Carmosino, Gao, Impagliazzo, Mihajlin, Paturi, Schneider [5]
- Williams [30]
- Björklund, K. [3, 15]

- In what follows we look at Williams's [30] proof system for #CNFSAT ...

Boolean satisfiability

- Let $x_1, x_2, \ldots, x_n$ be n variables that take values in $\{0, 1\}$
- A **truth assignment** A is a mapping that assigns a value in $\{0, 1\}$ to each of the variables $x_1, x_2, \ldots, x_n$
- A **literal** is a variable (x_i) or its negation ($\bar{x}_i$)
- A literal x_i (respectively, $\bar{x}_i$) is **satisfied** by A if $A(x_i) = 1$ (respectively, $A(x_i) = 0$)
- A **clause** C is a set of literals
- A clause C is **satisfied** by A if at least one literal in C is satisfied by A
- A collection of clauses $C_1, C_2, \ldots, C_m$ is **satisfied** by A if A satisfies every clause $C_1, C_2, \ldots, C_m$

Conjunctive-normal-form satisfiability (CNFSAT)

- The **CNFSAT** problem asks, given a collection $C_1, C_2, \ldots, C_m$ of clauses over variables $x_1, x_2, \ldots, x_n$ as input, whether there exists a truth assignment that satisfies $C_1, C_2, \ldots, C_m$
- CNFSAT is NP-complete
- The **#CNFSAT** problem asks, given a collection $C_1, C_2, \ldots, C_m$ of clauses over variables $x_1, x_2, \ldots, x_n$ as input, for the number of truth assignments that satisfy $C_1, C_2, \ldots, C_m$
- #CNFSAT is #P-complete
- It is not known how to solve CNFSAT in worst-case time $O^*((2-\epsilon)^n)$ for any constant $\epsilon > 0$; the best known algorithms run in $O^*(2^n)$ time
- Here the $O^*(\ )$ notation suppresses a multiplicative factor polynomial in the size of the input

CNFSAT and #CNFSAT

- It is easy to convince a verifier that an instance $C_1, C_2, \ldots, C_m$ of CNFSAT is satisfiable – just give the verifier a truth assignment A that satisfies $C_1, C_2, \ldots, C_m$
- The verifier can check that A actually satisfies $C_1, C_2, \ldots, C_m$ in time $O(mn)$
- But how to convince a verifier that $C_1, C_2, \ldots, C_m$ has exactly N satisfying truth assignments?
- For example, how to convince a verifier that $C_1, C_2, \ldots, C_m$ has *no* (zero) satisfying truth assignments?

A probabilistic proof system for #CNFSAT

- Williams's (2016) [30]:
  There exists a randomized algorithm V (the verifier) such that for all collections $\mathscr{C}$ of m clauses over n variables and all integers N it holds that
  1. if $\mathscr{C}$ has exactly N satisfying truth assignments, then there exists a bit string Π of length $O^*(2^{n/2})$ such that V accepts the triple $\mathscr{C}, N, \Pi$ with probability 1;
  2. if $\mathscr{C}$ does not have exactly N satisfying truth assignments, then for every bit string $\tilde{\Pi}$ it holds that V rejects the triple $\mathscr{C}, N, \tilde{\Pi}$ with probability $1 - o(1)$.

  Moreover, V runs in time $O^*(2^{n/2})$

Multivariate polynomial representation

- Let us work over $\mathbb{F}_q$, a finite field with $q \geq 2$ elements, q prime
- Let $x_1, x_2, \ldots, x_n$ be indeterminates that take values in $\mathbb{F}_q$
- Let us work with multivariate polynomials in $\mathbb{F}_q[x_1, x_2, \ldots, x_n]$
- We will transform a collection $\mathscr{C}$ of m clauses over $x_1, x_2, \ldots, x_n$ into a multivariate polynomial $p_{\mathscr{C}}(x_1, x_2, \ldots, x_n)$ such that for all $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\} \subseteq \mathbb{F}_q$ we have $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 1$ if and only if the truth assignment A with $A(x_1) = \alpha_1, A(x_2) = \alpha_2, \ldots, A(x_n) = \alpha_n$ satisfies $\mathscr{C}$, and $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$ otherwise

A literal as a multivariate polynomial

- For a literal ℓ over the variables $x_1, x_2, \ldots, x_n$, define the multivariate polynomial
$$p_\ell(x_1, x_2, \ldots, x_n) = \begin{cases} 1 - x_i & \text{if } \ell = x_i; \\ x_i & \text{if } \ell = \bar{x}_i \end{cases}$$
- p_ℓ has degree 1
- For all $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\}$ we have $p_\ell(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$ if and only if the truth assignment A with $A(x_1) = \alpha_1, A(x_2) = \alpha_2, \ldots, A(x_n) = \alpha_n$ satisfies ℓ, and $p_\ell(\alpha_1, \alpha_2, \ldots, \alpha_n) = 1$ otherwise

A clause as a multivariate polynomial

- Let C be a clause over the variables $x_1, x_2, \ldots, x_n$
- For a clause C, define the multivariate polynomial
$$p_C(x_1, x_2, \ldots, x_n) = 1 - \prod_{\ell \in C} p_\ell(x_1, x_2, \ldots, x_n)$$
- Since C has at most $2n$ literals, p_C has degree at most $2n$
- For all $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\}$ we have $p_C(\alpha_1, \alpha_2, \ldots, \alpha_n) = 1$ if and only if the truth assignment A with $A(x_1) = \alpha_1, A(x_2) = \alpha_2, \ldots, A(x_n) = \alpha_n$ satisfies C, and $p_C(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$ otherwise

A collection of clauses as a multivariate polynomial

- Let $\mathscr{C}$ be a collection $C_1, C_2, \ldots, C_m$ of clauses over the variables $x_1, x_2, \ldots, x_n$
- Define the multivariate polynomial

$$p_{\mathscr{C}}(x_1, x_2, \ldots, x_n) = \prod_{j=1}^{m} p_{C_j}(x_1, x_2, \ldots, x_n)$$

- $p_{\mathscr{C}}$ has degree at most $2mn$
- For all $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\}$ we have $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 1$ if and only if the truth assignment A with $A(x_1) = \alpha_1, A(x_2) = \alpha_2, \ldots, A(x_n) = \alpha_n$ satisfies $\mathscr{C}$, and $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$ otherwise

#CNFSAT as a multivariate polynomial

- Let us work over $\mathbb{F}_q$, a finite field with $q \geq 2$ elements, q a prime
- Let $x_1, x_2, \ldots, x_n$ be indeterminates that take values in $\mathbb{F}_q$
- Let $\mathscr{C}$ be a collection of m clauses over $x_1, x_2, \ldots, x_n$
- We now have a multivariate polynomial $p_{\mathscr{C}}(x_1, x_2, \ldots, x_n)$ of degree at most $2mn$ such that for all $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\}$ we have $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 1$ if and only if the truth assignment A with $A(x_1) = \alpha_1, A(x_2) = \alpha_2, \ldots, A(x_n) = \alpha_n$ satisfies $\mathscr{C}$, and $p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$ otherwise
- That is, the number N of satisfying truth assignments to $\mathscr{C}$ satisfies

$$N \equiv \sum_{\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0,1\}} p_{\mathscr{C}}(\alpha_1, \alpha_2, \ldots, \alpha_n) \pmod{q}$$

#CNFSAT as a univariate polynomial (1/2)

- Without loss of generality we may assume that n is even
- With some foresight, let us now assume that $2^{n/2+2}mn \leq q \leq 2^{n/2+3}mn$ (for large enough n we can find the two smallest such primes q_1, q_2 in time $O^*(2^{n/2})$, cf. [2] and [1])
- Let $a_1, a_2, \ldots, a_{n/2} \in \mathbb{F}_q[x]$ be univariate polynomials of degree at most $2^{n/2} - 1$ such that

$$\{0,1\}^{n/2} = \{(a_1(\alpha), a_2(\alpha), \ldots, a_{n/2}(\alpha)) : \alpha \in \{0, 1, \ldots, 2^{n/2} - 1\}\}$$

- In particular we can construct such polynomials $a_1, a_2, \ldots, a_{n/2}$ in time $O^*(2^{n/2})$ using fast interpolation (exercise)
- Now define the univariate polynomial $P_{\mathscr{C}} \in \mathbb{F}_q[x]$ in the indeterminate x by

$$P_{\mathscr{C}}(x) = \sum_{\alpha_{n/2+1}, \alpha_{n/2+2}, \ldots, \alpha_n \in \{0,1\}} p_{\mathscr{C}}\Big(a_1(x), a_2(x), \ldots, a_{n/2}(x), \alpha_{n/2+1}, \alpha_{n/2+2}, \ldots, \alpha_n\Big)$$

#CNFSAT as a univariate polynomial (2/2)

- Recalling from the previous slide, we have

$$P_{\mathscr{C}}(x) = \sum_{\alpha_{n/2+1},\alpha_{n/2+2},\ldots,\alpha_n\in\{0,1\}} p_{\mathscr{C}}\Big(a_1(x), a_2(x), \ldots, a_{n/2}(x), \alpha_{n/2+1}, \alpha_{n/2+2}, \ldots, \alpha_n\Big)$$

- We observe that $P_{\mathscr{C}}$ has degree at most $2^{n/2+1}mn \leq q/2$
- Using near-linear-time algorithms for univariate polynomials, given a collection $\mathscr{C}$ of clauses and a point $\xi \in \mathbb{F}_q$ as input, we can compute the value $P_{\mathscr{C}}(\xi)$ in time $O^*(2^{n/2})$ (exercise)
- From the definition of the polynomials $a_1, a_2, \ldots, a_{n/2}$ we observe that the number N of satisfying truth assignments to $\mathscr{C}$ satisfies

$$N \equiv \sum_{\alpha=0}^{2^{n/2}-1} P_{\mathscr{C}}(\alpha) \pmod{q} \tag{32}$$

The proof string

- Recall that for large enough n we can assume that we work modulo a prime q with $2^{n/2+2}mn \leq q \leq 2^{n/2+3}mn$
- Given $\mathscr{C}$ as input, in time $O^*(2^{n/2}e)$ we can produce e evaluations of $P_{\mathscr{C}}$ at distinct points
- If $e \geq 2^{n/2+1}mn + 1$, these evaluations enable us to interpolate $P_{\mathscr{C}}$ in time $O^*(2^{n/2})$ using fast interpolation
- We can represent the prime q and the coefficients of $P_{\mathscr{C}} \in \mathbb{F}_q[x]$ (of degree at most $2^{n/2+1}mn$) as a (prefix-coded) binary string Π_q of length $O^*(2^{n/2})$
- Let q_1, q_2 be the two least primes in the interval $[2^{n/2+2}mn, 2^{n/2+3}mn]$
- Take as the proof string Π the concatenation of Π_{q_1} and Π_{q_2}

Completeness

- Suppose $\Pi = \Pi_{q_1}\Pi_{q_2}$ is a correct proof string (of length $O^*(2^{n/2})$)
- Using Π_{q_1} and Π_{q_2} together with fast batch evaluation and (32) we can recover $N \bmod q_1$ and $N \bmod q_2$ in time $O^*(2^{n/2})$, where N is the number of satisfying truth assignments to $\mathscr{C}$
- Since $0 \le N \le 2^n$ and $q_1 q_2 \ge 2^n + 1$, from $N \bmod q_1$ and $N \bmod q_2$ we can reconstruct the correct N using the Chinese Remainder Theorem
- Thus the verifier will always accept a correct triple $\mathscr{C}, \tilde{N}, \tilde{\Pi}$ with $\tilde{\Pi} = \Pi$ and $\tilde{N} = N$ in time $O^*(2^{n/2})$

Soundness (probabilistic) I

- Suppose the verifier is given as input a collection $\mathscr{C}$ of m clauses over the variables $x_1, x_2, \ldots, x_n$, an integer $\tilde{N}$, and a binary string $\tilde{\Pi}$
- The verifier first checks that $\tilde{\Pi} = \tilde{\Pi}_{q_1}\tilde{\Pi}_{q_2}$ such that $\tilde{\Pi}_{q_1}$ and $\tilde{\Pi}_{q_2}$ encode the coefficients of a polynomial $\tilde{P}$ of degree at most $2^{n/2+1}mn$ modulo the two least primes q_1 and q_2 in the interval $[2^{n/2+2}mn, 2^{n/2+3}mn]$; if this is not the case, the verifier rejects
- Next, consider each $q \in \{q_1, q_2\}$ in turn
- To verify that $\tilde{P} = P_{\mathscr{C}} \in \mathbb{F}_q[x]$ the verifier repeats the following test $\lceil \log_2 n \rceil + 1$ times: select $\xi \in \mathbb{F}_q$ independently and uniformly at random, and test that $\tilde{P}(\xi) = P_{\mathscr{C}}(\xi)$ holds; if this is not the case, the verifier rejects
- The left-hand side $\tilde{P}(\xi)$ can be evaluated in time $O^*(2^{n/2})$ using Horner's rule; the right-hand side $P_{\mathscr{C}}(\xi)$ can be evaluated in time $O^*(2^{n/2})$ using the dedicated evaluation algorithm for $P_{\mathscr{C}}$ (in the exercises)

Soundness (probabilistic) II

- Since $\tilde{P} - P_{\mathscr{C}}$ has degree at most $2^{n+1}mn \leq q/2$, if $\tilde{P} \neq P_{\mathscr{C}} \in \mathbb{F}_q[x]$ then the verifier rejects with probability at least $1 - 1/n$ (exercise)
- Thus the verifier rejects with probability $1 - o(1)$ unless the string $\tilde{\Pi}$ is in fact the correct proof string Π; from Π the verifier can recover the correct solution N and reject unless $\tilde{N} = N$; the verifier runs in time $O^*(2^{n/2})$

Complexity of preparing and verifying the proof

- Given $\mathscr{C}$ as input, in time $O^*(2^{n/2}e)$ we can produce e evaluations of $P_{\mathscr{C}}$ at distinct points modulo q
- If $e \geq 2^{n/2+1}mn + 1$, these evaluations enable us to interpolate $P_{\mathscr{C}}$ in time $O^*(2^{n/2})$ using fast interpolation
- Thus, the total effort to prepare the proof is $O^*(2^n)$, which essentially matches the best known algorithms for counting the number of satisfying assignments to $\mathbb{C}$ (that is, no algorithm that runs in worst-case time $O^*((2-\epsilon)^n)$ is known for any constant $\epsilon > 0$)
- The total effort to (probabilistically) verify the proof is $O^*(2^{n/2})$

Proof preparation with tolerance for errors [3, 15]

- Beyond #CNFSAT, a number of other computational problems admit proof systems in the following framework ...
- The proof is a polynomial $p(x)$ of degree at most d over $\mathbb{F}_q$ (one or more polynomials with Chinese Remaindering)
- **Prepare** the proof in **evaluation** representation with distinct e points

$$(\xi_1, p(\xi_1)),\ (\xi_2, p(\xi_2)),\ \ldots,\ (\xi_e, p(\xi_e))$$

- Preparation is vector-parallel, tolerates at most $(e-d-1)/2$ errors for $e \geq d+1$
- **Decode** the proof from evaluation representation to **coefficient** representation

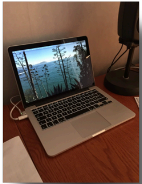
$$p(x) = \pi_0 + \pi_1 x + \pi_2 x^2 + \ldots + \pi_d x^d$$

- **Verify** the proof by selecting a uniform random $\xi \in \mathbb{F}_q$ and testing whether

$$p(\xi) = \pi_0 + \pi_1 \xi + \pi_2 \xi^2 + \ldots + \pi_d \xi^d$$

Delegating computation

Client



modest resources
reliable

Problem instance

Solution

Service-provider

massively SIMD-parallel resources
error-prone

- How to verify that the solution is correct ?

- How to design an algorithm to tolerate (a small number of) errors *during computation* ?

- How to convince the client or a third party that the solution is correct ?

Recap of Lecture 6

- We look at yet further applications of the evaluation–interpolation duality and randomization in algorithm design
- Randomized **identity testing** for polynomials and matrices (exercise)
- **Delegating computation** and **proof systems**
- **Completeness** and **soundness** of a proof system,
  cost of **preparing** a proof, cost of **verifying** a proof
- Williams's (2016) [30] probabilistic proof system for #CNFSAT
- Coping with **errors in computation** using error-correcting codes with multiplicative structure (Reed–Solomon codes revisited)
- Proof systems that tolerate errors during proof preparation (Björklund & K. 2016) [3]
- An extension of Shamir's secret sharing to delegating a computation to multiple counterparties (delegating matrix multiplication, exercise)

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic, online, and randomised algorithms
- Ways of coping with uncertainty in computation, including error-correction and proofs of correctness
- The art of solving a large problem by reduction to one or more smaller instances of the same or a related problem
- (Linear) independence, dependence, and their abstractions as enablers of efficient algorithms

Learning objectives (2/2)

- Making use of duality
  - Often a problem has a corresponding **dual** problem that is obtainable from the original (the **primal**) problem by means of an easy transformation
  - The primal and dual control each other, enabling an algorithm designer to use the interplay between the two representations
- Relaxation and tradeoffs between objectives and resources as design tools
  - Instead of computing the exact optimum solution at considerable cost, often a less costly but principled approximation suffices
  - Instead of the complete dual, often only a randomly chosen partial dual or other relaxation suffices to arrive at a solution with high probability