Error Correction in Fast Matrix Multiplication and Inverse

Daniel S. Roche

Computer Science Department
United States Naval Academy
Annapolis, Maryland, U.S.A.

ISSAC 2018
19 July 2018
New York City

Correcting Errors

General Framework

- Some problem is slow to solve locally.
- So we outsource the computation and get back the result.
- If the result is wrong, we should detect that.
- If the result is “close”, we should correct it.

Sources of error

- Noisy communication channel
- Faults in computation
- Malicious servers?

Example: Sudoku

Sent to server

	7		8		2			
								7
1	4	8	3	5				
		5			8	3		
	6							9
3						7		
5						1	7	
		4		9				
		7	4					2

Response

9	7	6	8	1	2	4	5	3
2	5	3	9	6	4	8	1	7
1	4	5	3	8	7	9	2	6
7	9	8	6	2	5	3	4	1
4	6	1	5	7	3	2	8	9
3	8	2	1	4	9	7	6	5
5	3	9	2	9	6	1	7	4
6	2	4	7	9	1	5	3	8
8	1	7	4	3	8	6	9	2

Source: `https://www.nytimes.com/crosswords/game/sudoku/hard`

Example: Sudoku

Sent to server

	7		8		2			
								7
1	4	8	3	5				
		5			8	3		
	6							9
3						7		
5						1	7	
		4		9				
		7	4					2

Response

9	7	6	8	1	2	4	5	3
2	5	3	9	6	4	8	1	7
1	4	5	3	8	7	9	2	6
7	9	8	6	2	5	3	4	1
4	6	1	5	7	3	2	8	9
3	8	2	1	4	9	7	6	5
5	3	9	2	9	6	1	7	4
6	2	4	7	9	1	5	3	8
8	1	7	4	3	8	6	9	2

Source: `https://www.nytimes.com/crosswords/game/sudoku/hard`

First Problem: Matrix Multiplication

Matrix Multiplication with Errors

Input: $A, B, C \in \mathsf{F}^{n \times n}$, where $C \approx AB$

Output: $E \in \mathsf{F}^{n \times n}$ s.t. $AB = C - E$

Parameters:

- Dimension n
- Input size $t = \#A + \#B + \#C$
- Error count $k = \#E$

First Problem: Matrix Multiplication

Matrix Multiplication with Errors

Input: $A, B, C \in \mathsf{F}^{n \times n}$, where $C \approx AB$

Output: $E \in \mathsf{F}^{n \times n}$ s.t. $AB = C - E$

Parameters:

- Dimension n
- Input size $t = \#A + \#B + \#C$
- Error count $k = \#E$

- Naïve recomputation: $O(n^\omega)$
- Gasieniec, Levcopoulos, Lingas, Pagh, Tokuyama, *Algorithmica* 2017: $\widetilde{O}\left(n^2 + kn\right)$

First Problem: Matrix Multiplication

Matrix Multiplication with Errors

Input: $A, B, C \in \mathsf{F}^{n \times n}$, where $C \approx AB$

Output: $E \in \mathsf{F}^{n \times n}$ s.t. $AB = C - E$

Parameters:

- Dimension n
- Input size $t = \#A + \#B + \#C$
- Error count $k = \#E$

- Naïve recomputation: $O(n^\omega)$
- Gasieniec, Levcopoulos, Lingas, Pagh, Tokuyama, *Algorithmica* 2017: $\widetilde{O}\left(n^2 + kn\right)$
- Ours: $\widetilde{O}(t + kn)$ worst-case; $\widetilde{O}\left(t + k^{0.686}n\right)$ best case

Related Work

Monte Carlo **verification** in Linear Algebra

- Matrix multiplication: Freivalds '79
- Positive semidefiniteness: Kaltofen, Nehring, Saunders '11
- Rank, characteristic polynomial,. . . : Dumas & Kaltofen '14
- Rank profile, triangular forms: Dumas, Lucas, Pernet '17

Error correction in symbolic computation

Complexity of sparse matrix multiplication

Related Work

Monte Carlo **verification** in Linear Algebra

Error correction in symbolic computation

- Chinese remaindering: Goldreich, Ron, Sudan '99
- Rational reconstruction: Khonji, Pernet, Roch, Roche, Stalinski '10
- Sparse interpolation: Comer, Kaltofen, Pernet '12

Complexity of sparse matrix multiplication

Related Work

Monte Carlo **verification** in Linear Algebra

Error correction in symbolic computation

Complexity of sparse matrix multiplication

k = number of nonzeros

- $\widetilde{O}\left(n^2 + k^{0.7} n^{1.2}\right)$: Yuster & Zwick '04
- $\widetilde{O}\left(k^{1.34}\right)$ output sensitive: Amossen & Pagh '09
- $\widetilde{O}\left(k^{0.188} n^2\right)$: Lingas '10
- $\widetilde{O}\left(n^2 + kn\right)$ output sensitive: Pagh '13

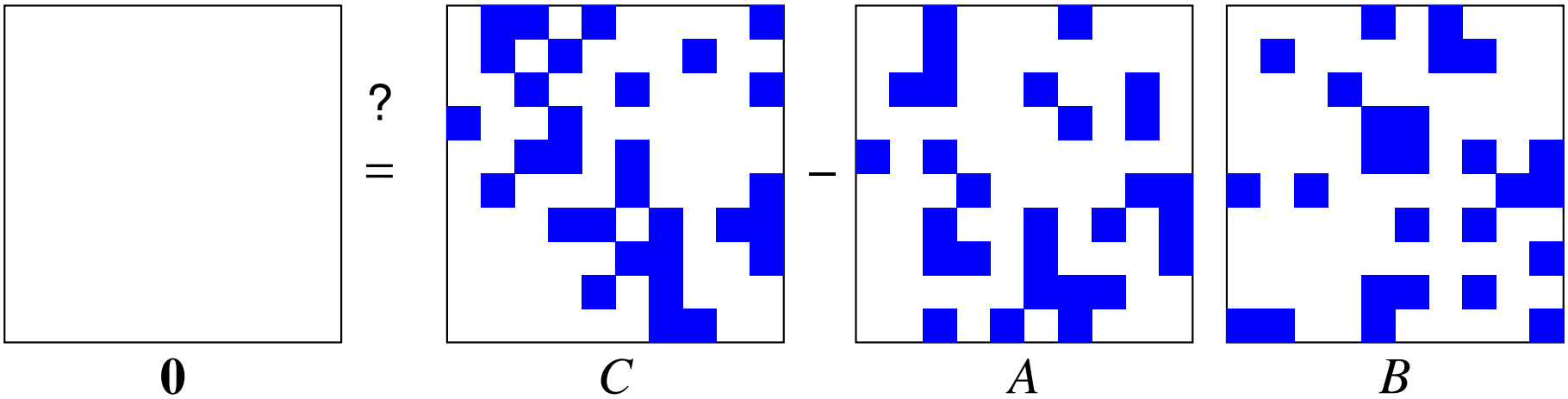
Outline

- Introduction
- Matrix Multiplication with Errors
  Crucial tools: Nonzero row identification, Sparse interpolation
- Matrix Inverse with Errors
  Additional tools: Commutativity, Sparse low-rank linear algebra
- Preliminary implementation timings

Detecting errors

R. Freivalds: Fast Probabilistic Algorithms (1979)
Given A, B, C, does $AB = C$?

1. Choose random vector v
2. Compute $(C - AB)v$
3. Check whether $(C - AB)v = \mathbf{0}$

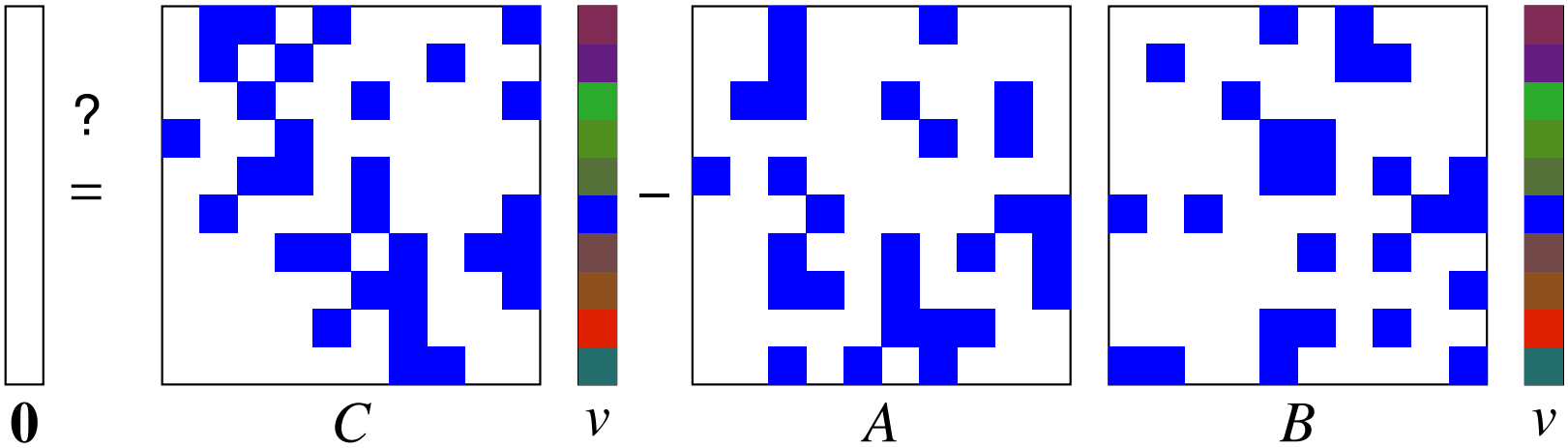


Detecting errors

R. Freivalds: Fast Probabilistic Algorithms (1979)

Given A, B, C, does $AB = C$?

1. Choose random vector v
2. Compute $(C - AB)v$
3. Check whether $(C - AB)v = \mathbf{0}$

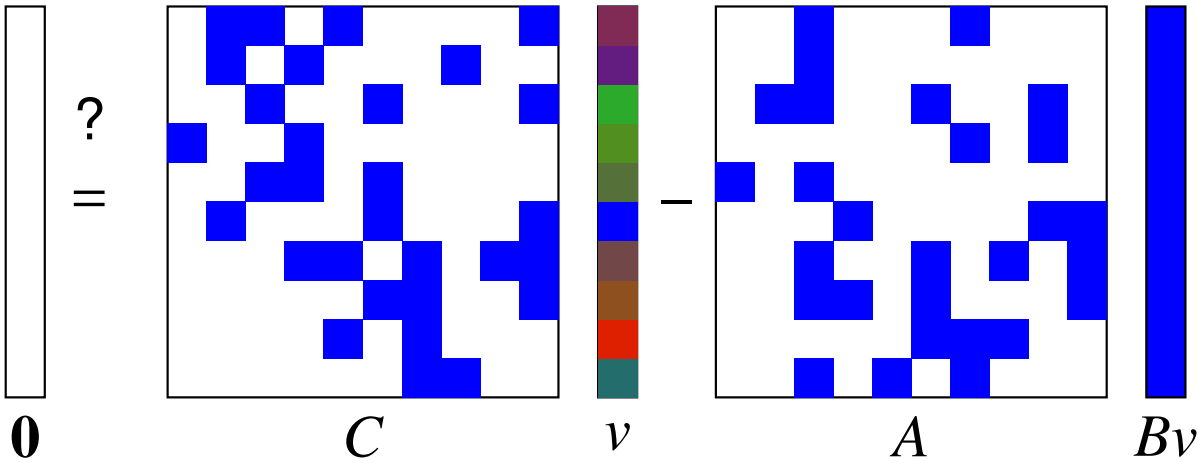


Detecting errors

R. Freivalds: Fast Probabilistic Algorithms (1979)

Given A, B, C, does $AB = C$?

1. Choose random vector v
2. Compute $(C - AB)v$
3. Check whether $(C - AB)v = \mathbf{0}$

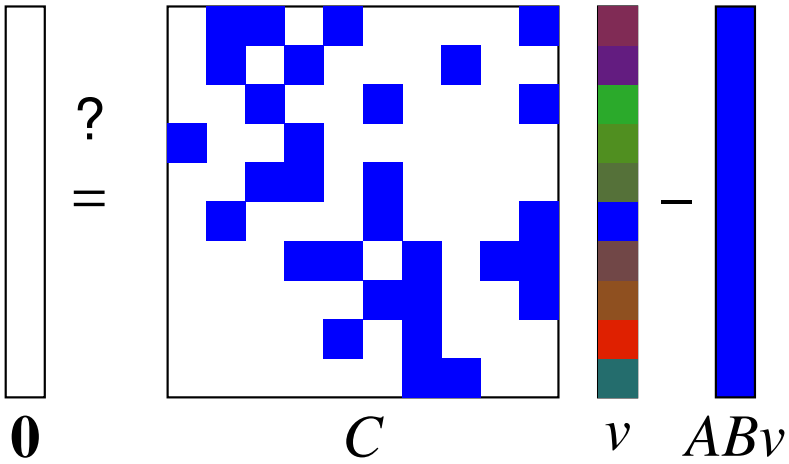


Detecting errors

R. Freivalds: Fast Probabilistic Algorithms (1979)

Given A, B, C, does $AB = C$?

1. Choose random vector v
2. Compute $(C - AB)v$
3. Check whether $(C - AB)v = \mathbf{0}$

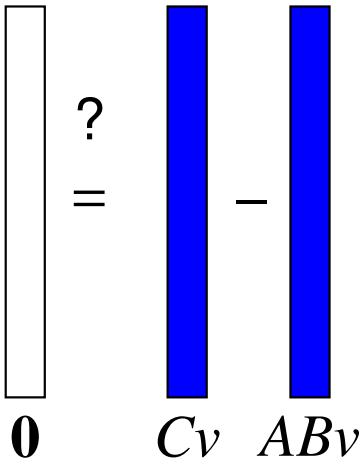


Detecting errors

R. Freivalds: Fast Probabilistic Algorithms (1979)

Given A, B, C, does $AB = C$?

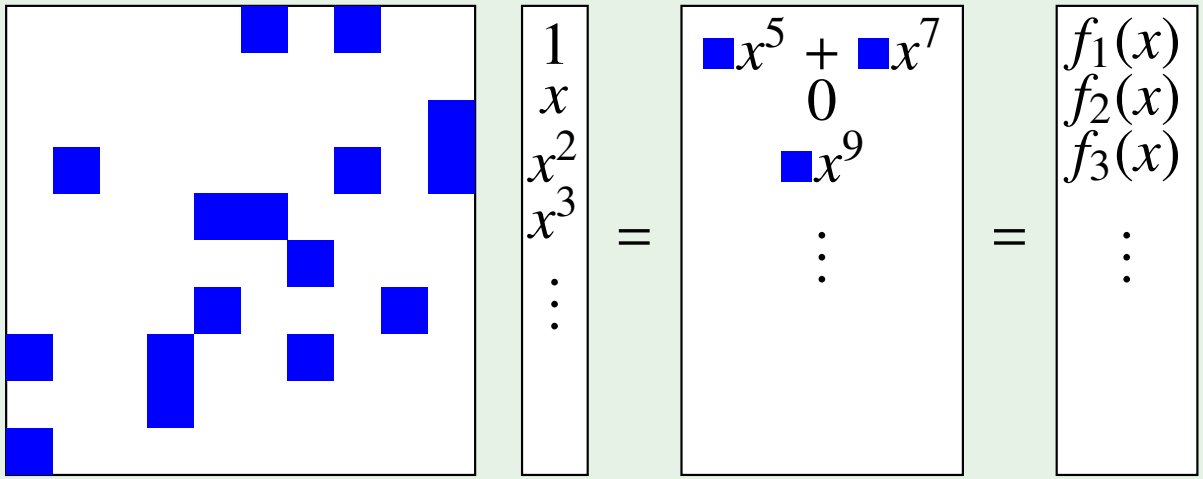
1. Choose random vector v
2. Compute $(C - AB)v$
3. Check whether $(C - AB)v = \mathbf{0}$



Row Polynomials

Treat each row of $M \in \mathsf{F}^{n \times n}$ as a degree-$(n-1)$ polynomial

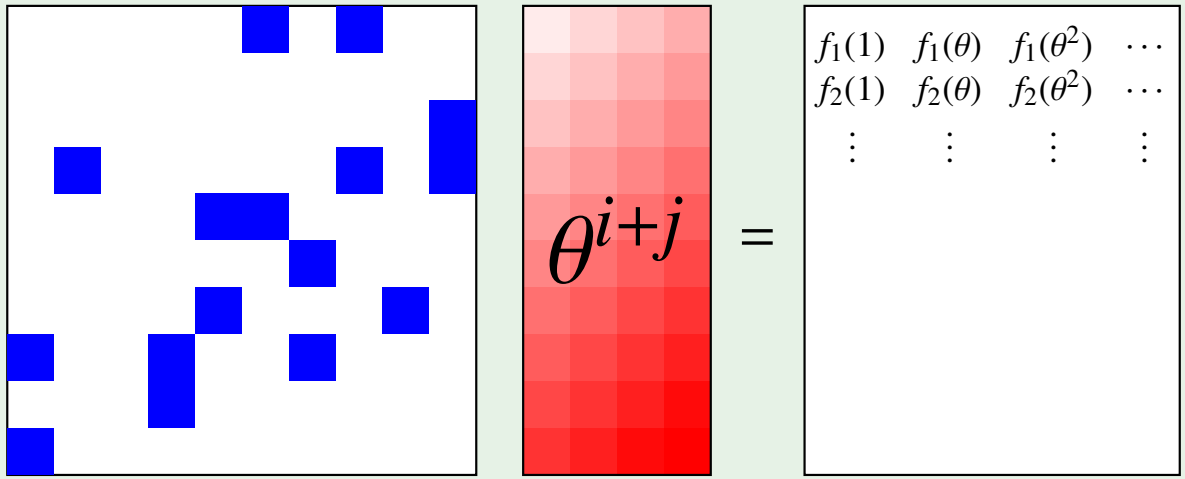
Example



Row Polynomial Evaluation

- Need to evaluate each row polynomial $f_i(x)$ at $1, \theta, \theta^2, \ldots$
- This is equivalent to multiplying by a (truncated) DFT matrix.

Example



M $n \times n$ | V $n \times s$ | MV $n \times s$

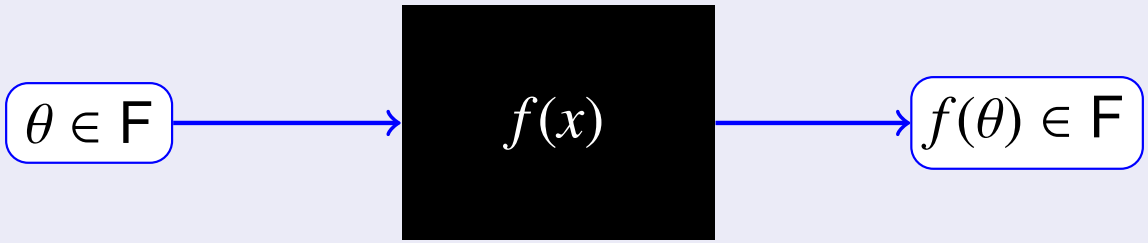
Important: Takes $\widetilde{O}(sn + t)$ field ops, where $t = \#M$

Sparse polynomial interpolation

Overview

Given a way to evaluate an unknown polynomial $f \in \mathsf{F}[x]$
with at most s nonzero terms,
how to recover the nonzero coefficients and exponents of f?

Polynomial black box



Sparse polynomial interpolation

Prony's Method

Prony's (1795) method from exponential analysis can be used here.
(Ben-Or & Tiwari '88, Kaltofen Lakshman & Wiley '91,...)

Algorithm Overview

1. Find a high-order element $\omega \in \mathsf{F}$
2. Compute $f(\omega^i)$ for $i = 0, 1, \ldots, 2s-1$
3. Berlekamp-Massey to find annihilator $\Lambda(z)$
4. Find roots ω^{e_i} of Λ
5. Compute discrete logs e_i of Λ
6. Transp. Vandermonde solve to get coefficients c_i

Sparse polynomial interpolation

Prony's Method

Prony's (1795) method from exponential analysis can be used here.
(Ben-Or & Tiwari '88, Kaltofen Lakshman & Wiley '91,...)

Algorithm Overview

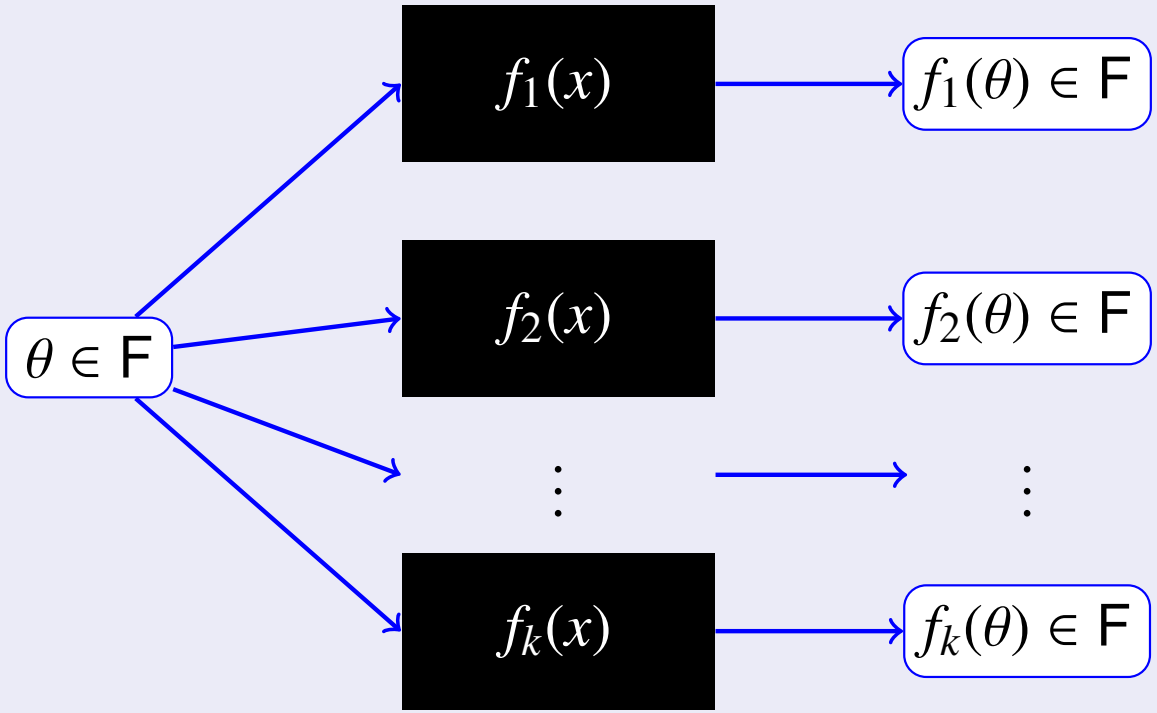
1. Find a high-order element $\omega \in \mathsf{F}$
2. Compute $f(\omega^i)$ for $i = 0, 1, \ldots, 2s-1$
3. Berlekamp-Massey to find annihilator $\Lambda(z)$
4. Find roots ω^{e_i} of Λ
5. Compute discrete logs e_i of Λ
6. Transp. Vandermonde solve to get coefficients c_i

- These steps depend on the field F
- Root finding and discrete logarithms are generally the bottleneck (especially over finite fields!)

Batched Sparse Polynomial Interpolation

What if we simultaneously interpolate k sparse polynomials?

Simultaneous polynomial black box



Batched Sparse Polynomial Interpolation

Algorithm

Challenge: Simultaneously interpolate $f_1, \ldots, f_k \in \mathsf{F}[x]$, each having $\leq s$ nonzero terms and degree $\leq n$.

Idea: Precomputation of

- High-order element $\omega \in \mathsf{F}$
- First n powers of ω

Result: Recovery in $\widetilde{O}(n + sk)$ time for any field F

Correcting errors in matrix multiplication

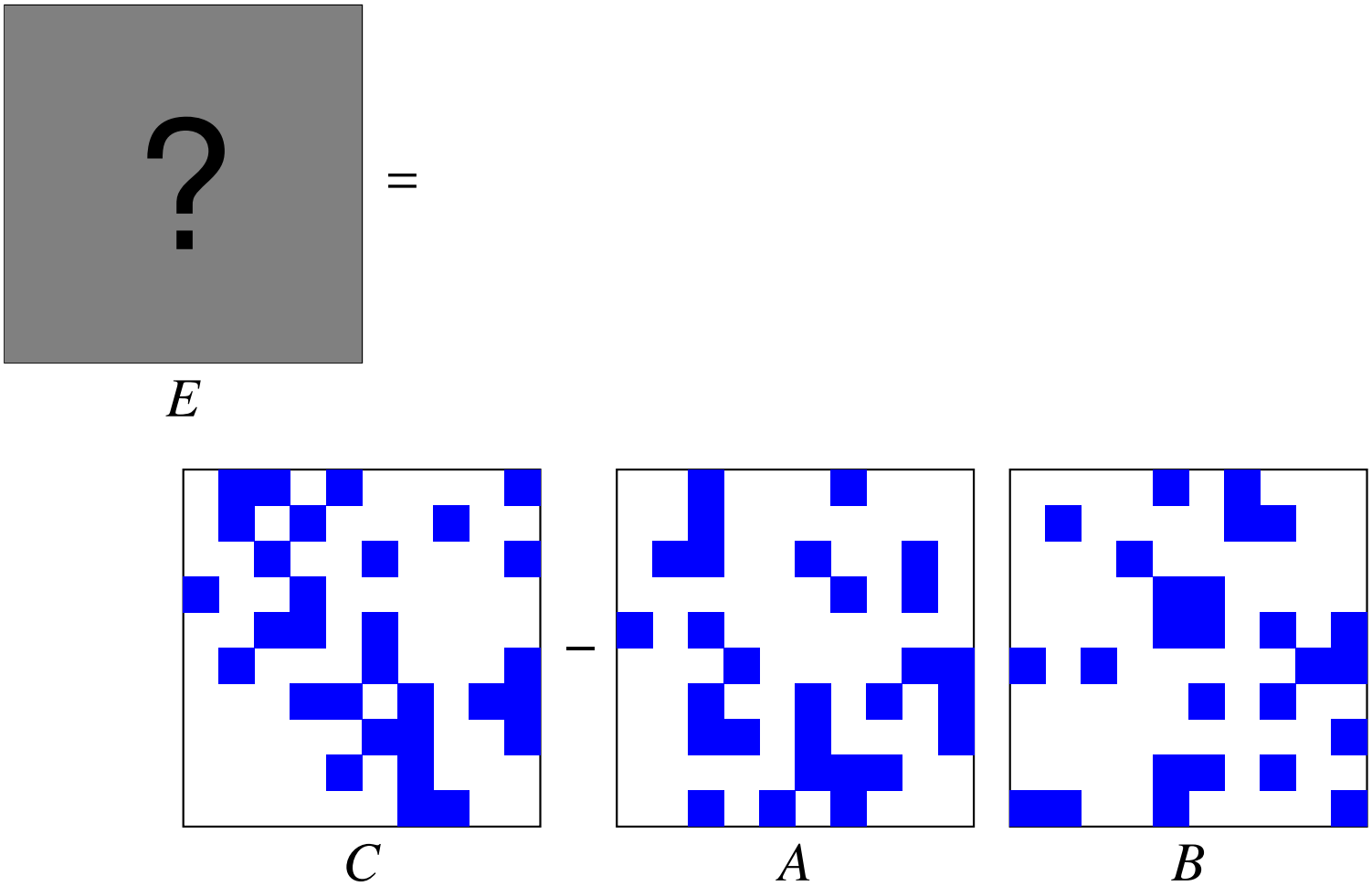
Input: $A, B, C \in \mathsf{F}^{n \times n}$
Output: $E \in \mathsf{F}^{n \times n}$ s.t. $AB = C - E$

Algorithm overview:

1. Formulate as $E = C - AB$
2. Find nonzero rows
   Choose random $v \in \mathsf{F}^n$ and compute Ev
3. Evaluate row polynomials
   For each evaluation θ, compute $E[1, \theta, \ldots, \theta^{n-1}]^T$
4. Interpolate row polynomials
   Sparse interpolation to recover at least half of the nonero rows
5. Repeat $O(\log k)$ times until all rows are found

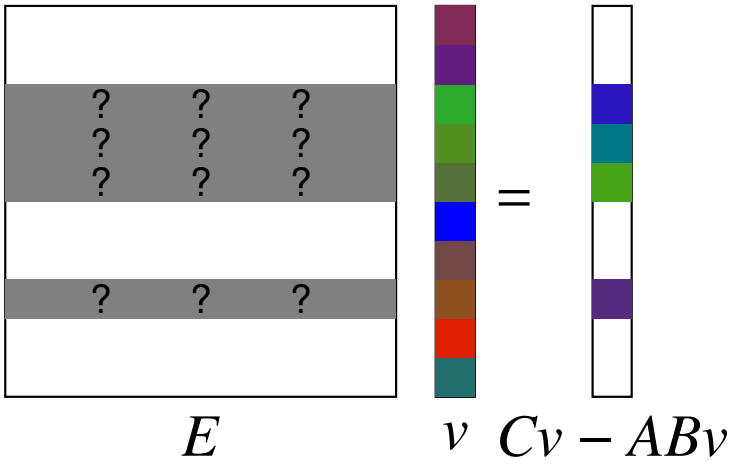
Multiplication Algorithm Overview

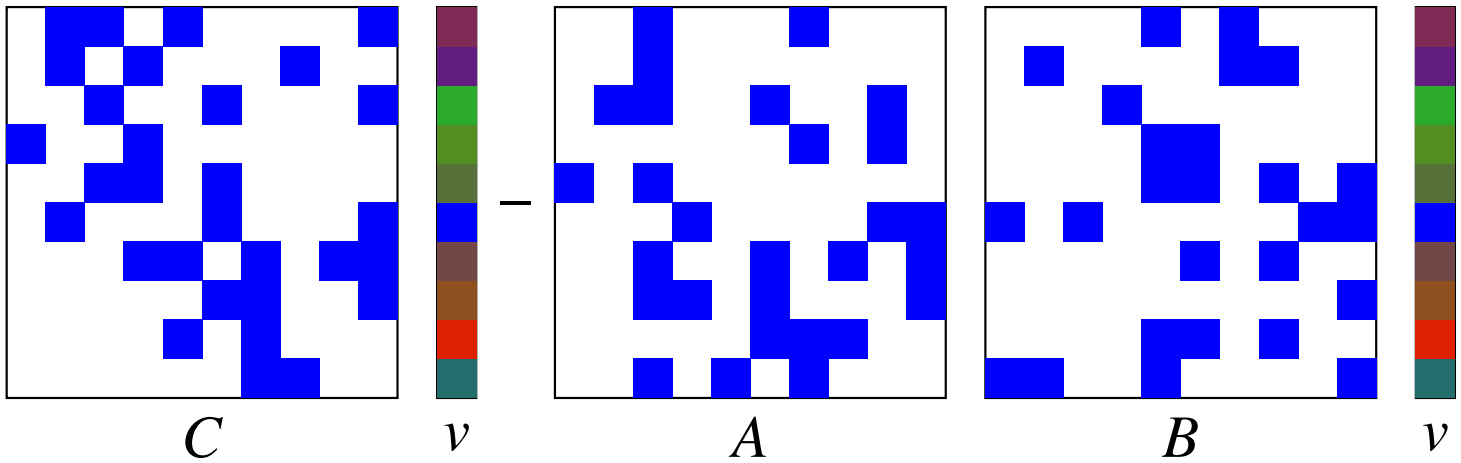
Given A, B, C, we want to compute $E = C - AB$



Multiplication Algorithm Overview

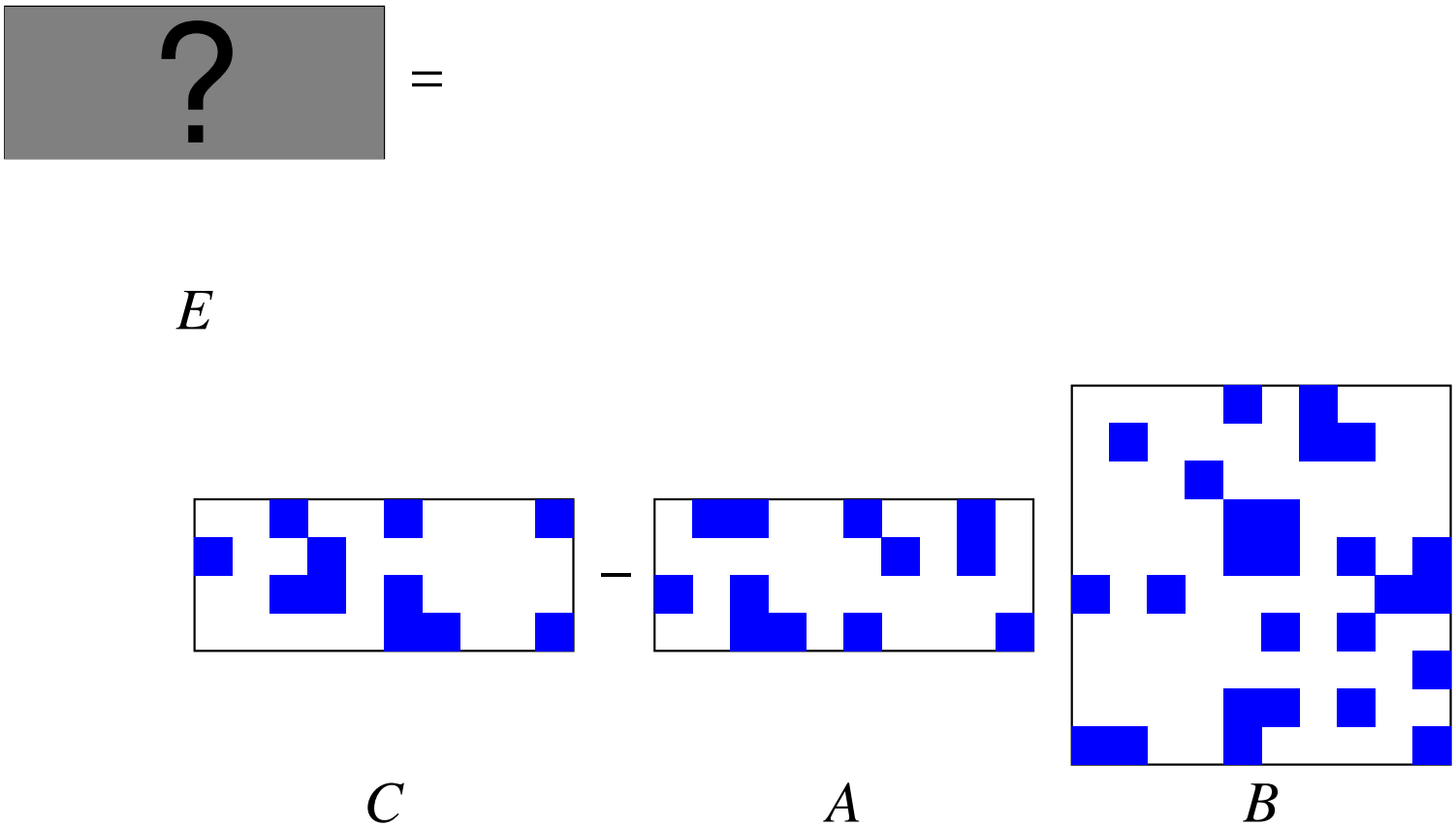
1 Find zero rows of E





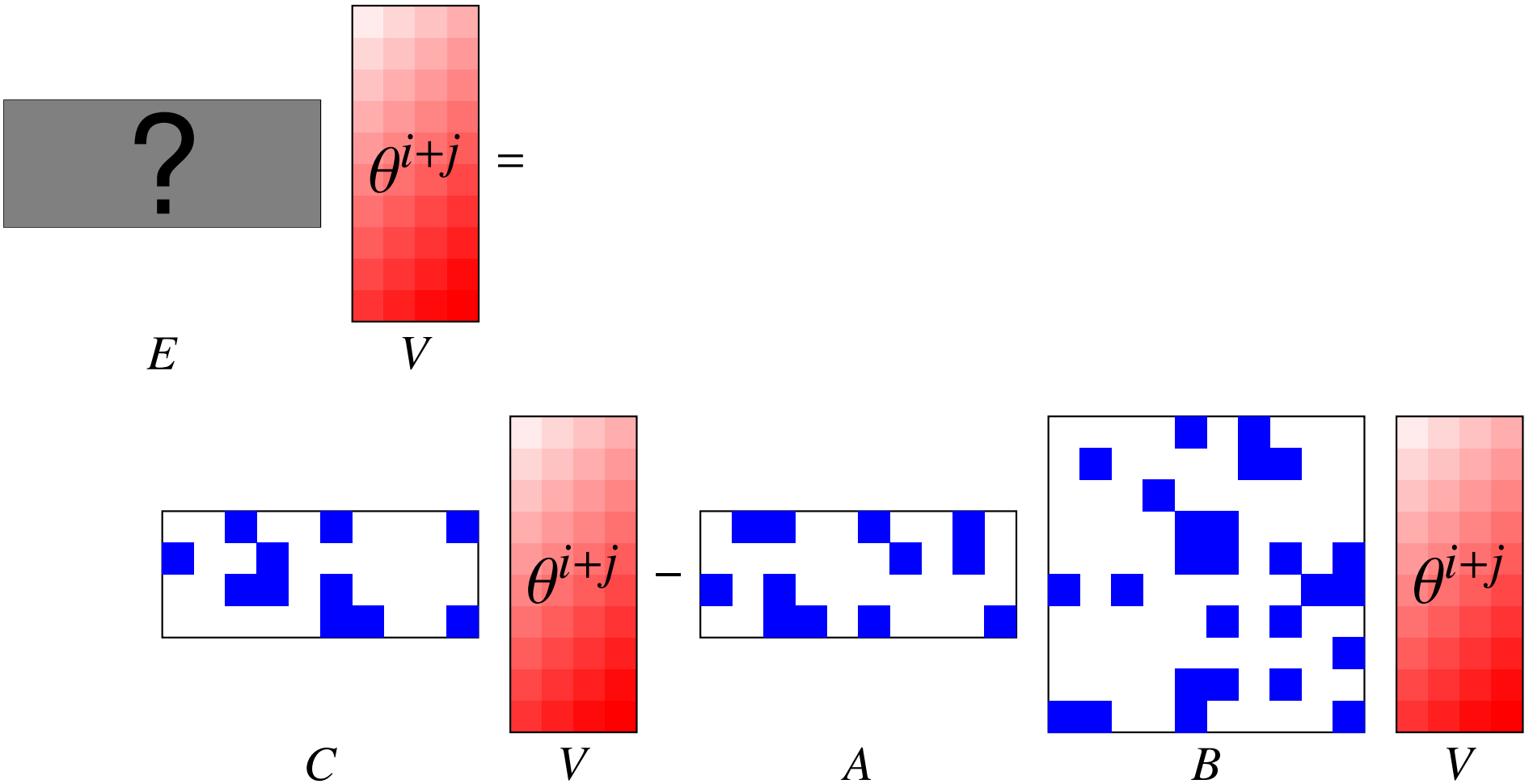
Multiplication Algorithm Overview

2 Remove corresponding rows from C and A



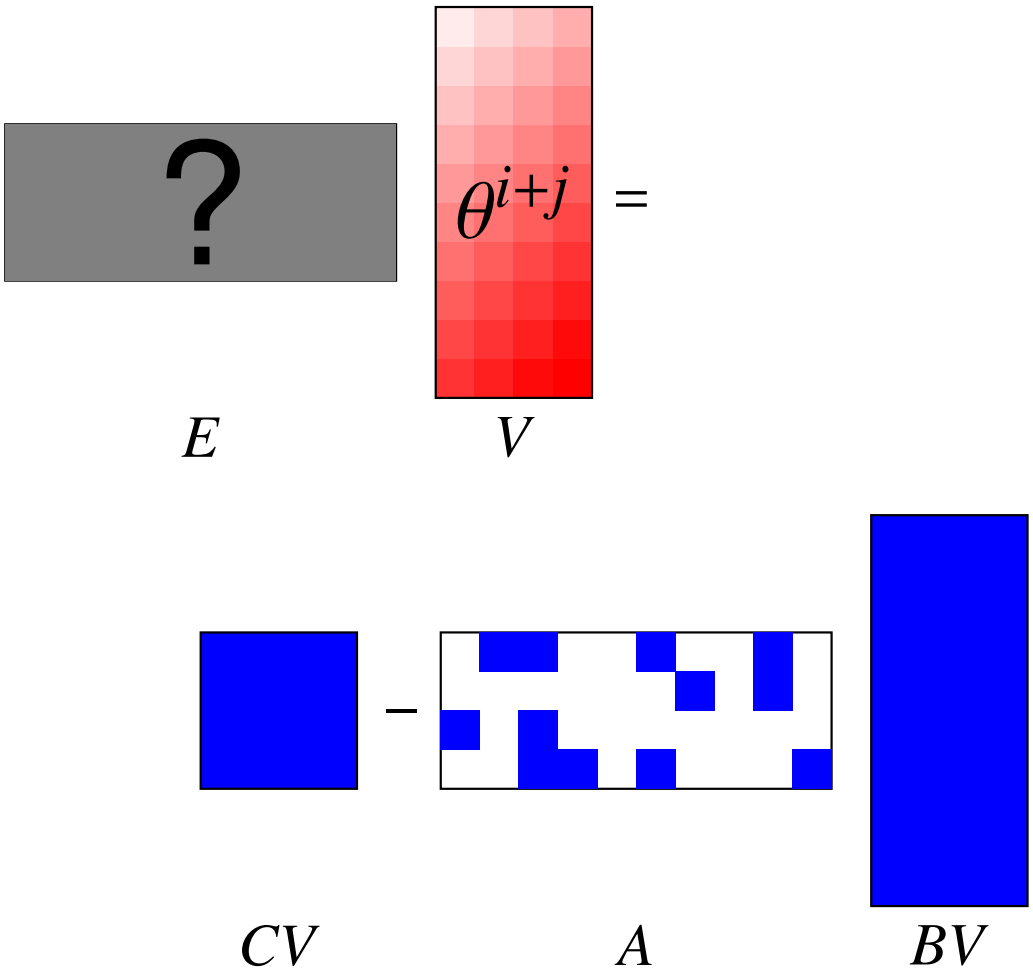
Multiplication Algorithm Overview

3 Choose $s \leq n$ and evaluate row polynomials at $2s$ powers of θ



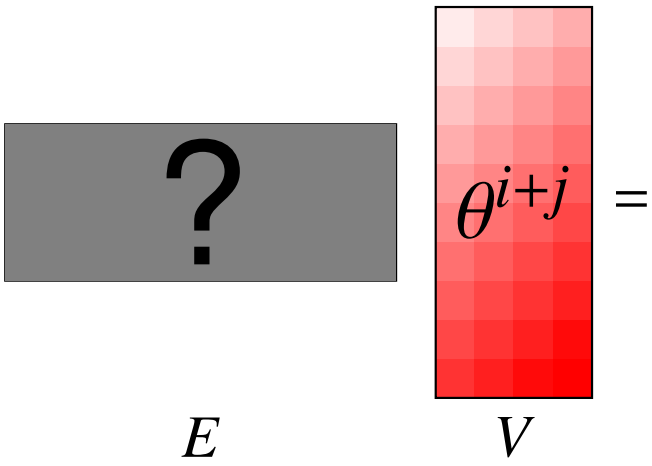
Multiplication Algorithm Overview

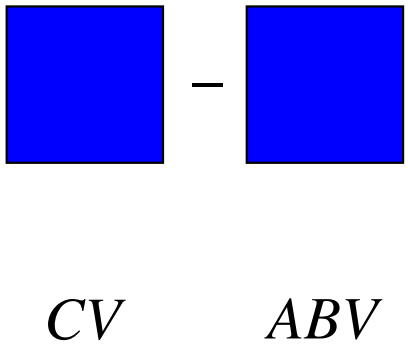
3 Choose $s \leq n$ and evaluate row polynomials at $2s$ powers of θ



Multiplication Algorithm Overview

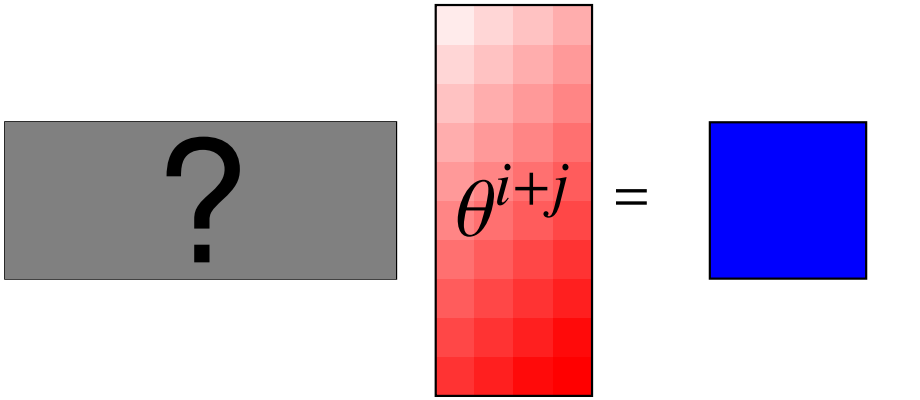
3 Choose $s \leq n$ and evaluate row polynomials at $2s$ powers of θ





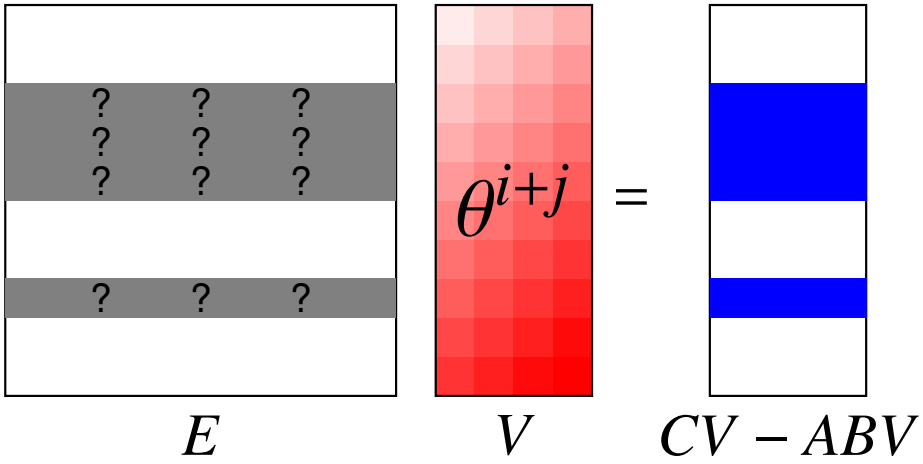
Multiplication Algorithm Overview

3 Choose $s \leq n$ and evaluate row polynomials at $2s$ powers of θ



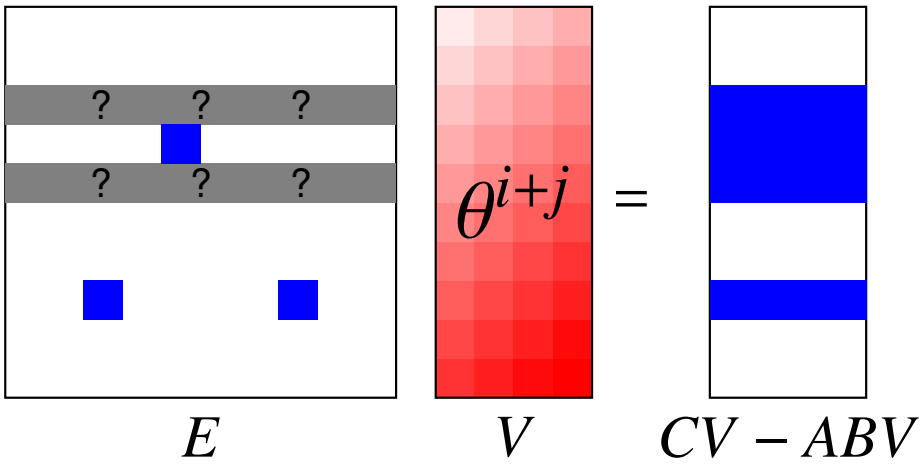
Multiplication Algorithm Overview

3 Choose $s \leq n$ and evaluate row polynomials at $2s$ powers of θ



Multiplication Algorithm Overview

4 Perform sparse interpolation to recover s-sparse rows of E



Choosing the number of evaluations

Each row polynomial $f_1, \ldots, f_r$ is evaluated at $\omega^0, \ldots, \omega^{2s-1}$
Question: How large should s be?

- To recover every row, need $s \geq \max_i \#f_i$
  Therefore s could be as large as $k - r = O(k)$

Choosing the number of evaluations

Each row polynomial $f_1, \ldots, f_r$ is evaluated at $\omega^0, \ldots, \omega^{2s-1}$
Question: How large should s be?

- To recover every row, need $s \geq \max_i \#f_i$
  Therefore s could be as large as $k - r = O(k)$
- To recover at least half of the rows, only need

$$s \geq \frac{2k}{r}$$

  Then sparse interpolation costs $\widetilde{O}(rs + n) = \widetilde{O}(k + n)$

Iterative updates

The problem can be transposed easily:

- $E = C - AB$
- $E^T = C^T - B^T A^T$

After each step, at least half of the rows are completely recovered.

We transpose at each step to minimize r.

Multiplication Complexity

Total cost in field operations is

$$\widetilde{O}\left(n + t + k \cdot \min\left(\frac{t}{r} + 1, \frac{n}{\min(r, \frac{k}{r})^{3-\omega}}\right)\right)$$

- $n =$ dimension
- $t =$ number of nonzeros in A, B, C
- $k =$ number of errors (nonzeros in E)
- $r =$ number of nonzero rows or columns in E, whichever is smaller
- $\omega < 2.373$, exponent of matrix multiplication

Multiplication Complexity

Total cost in field operations is $\widetilde{O}\left(n + t + k \min(\frac{t}{r} + 1, n/\min(r, \frac{k}{r})^{3-\omega})\right)$

- n = dimension
- t = number of nonzeros in A, B, C
- k = number of errors (nonzeros in E)
- r = number of nonzero rows or columns in E, whichever is smaller
- $\omega < 2.373$, exponent of matrix multiplication

Very sparse	"Compact" errors	Worst case	
$t \ll nk^{\omega/2}$	$r \approx \sqrt{k}$	$k \leq n$	$k \geq n$
$tk^{0.5}$	$t + k^{0.69}n$ $t + k^{(\omega-1)/2}n$	$t + kn$	$t + k^{0.38}n^{1.63}$ $t + k^{\omega-2}n^{4-\omega}$

Second Problem: Matrix Inverse

Matrix Inverse with Errors

Input: $A, B \in \mathsf{F}^{n \times n}$

Output: $E \in \mathsf{F}^{n \times n}$ s.t. $A^{-1} = B + E$

Parameters:

- Dimension n
- Input size $t = \#A + \#B$
- Error count $k = \#E$

- Naïve recomputation: $O(n^\omega)$
- Ours: $\widetilde{O}(t + kn + k^\omega)$ worst-case; $\widetilde{O}\left(t + k^{0.69} n\right)$ best case

Problem Formulation

Rewriting the problem definition:

$$EA = I - BA$$
$$AE = I - AB$$

Crucial steps in multiplication algorithm:

- Computing nonzero rows of E
  Evaluate $EAv = v - A(Bv)$ for random vector v
- Evaluating row polynomials at $2s$ powers of θ
  (Coming up next. . .)

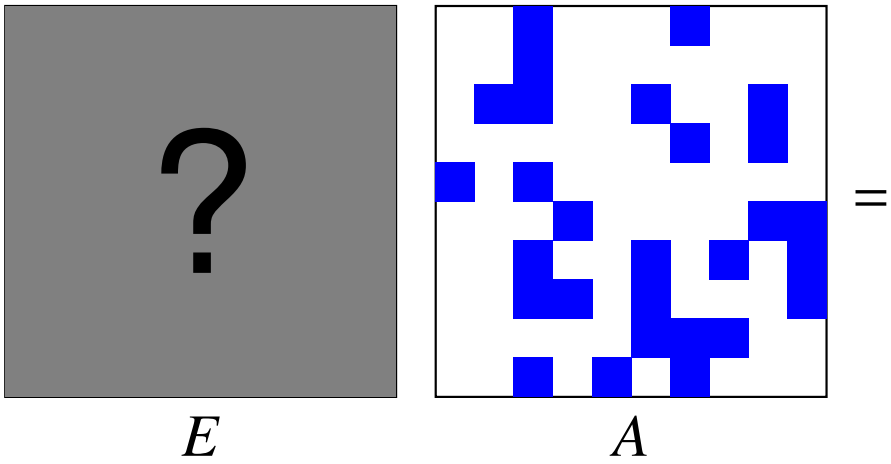
Low-Rank Linear Algebra

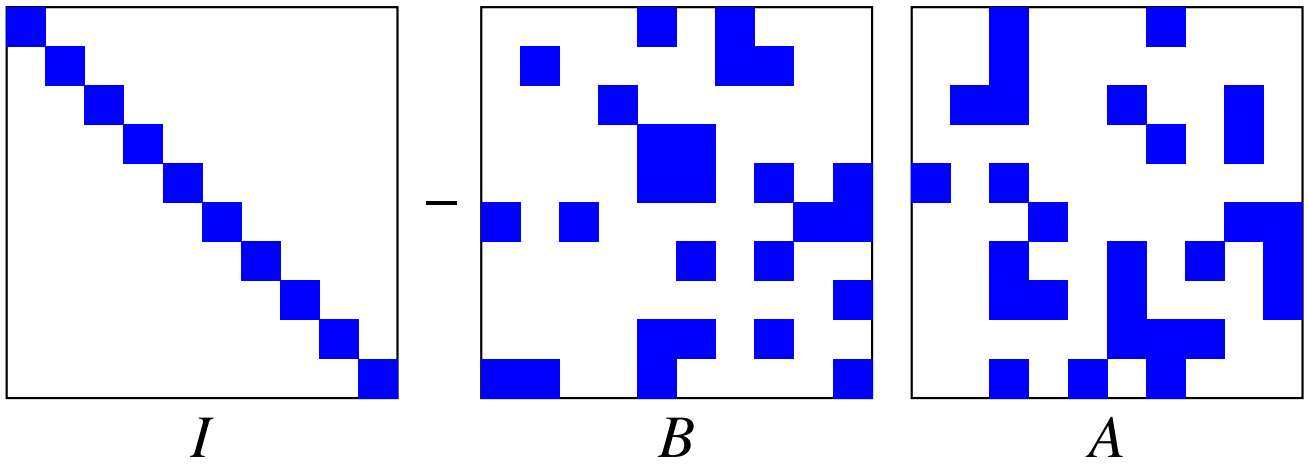
Algorithm (Chung, Kwok, Lau '13)

Input: Matrix M with rank r and t nonzeros
Output: Indices of r linearly independent rows
Cost: $\widetilde{O}(t + r^{\omega})$

Inverse Algorithm Overview

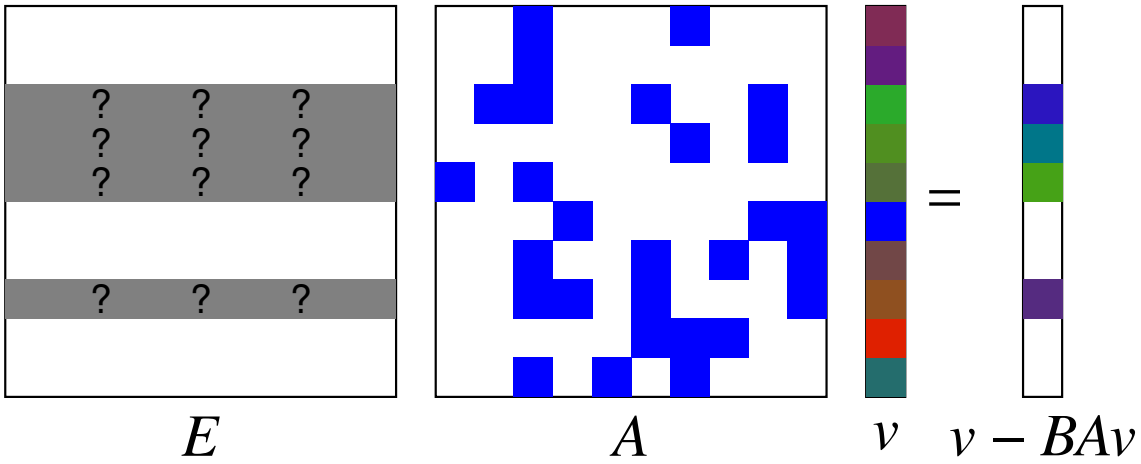
Write $EA = I - BA$

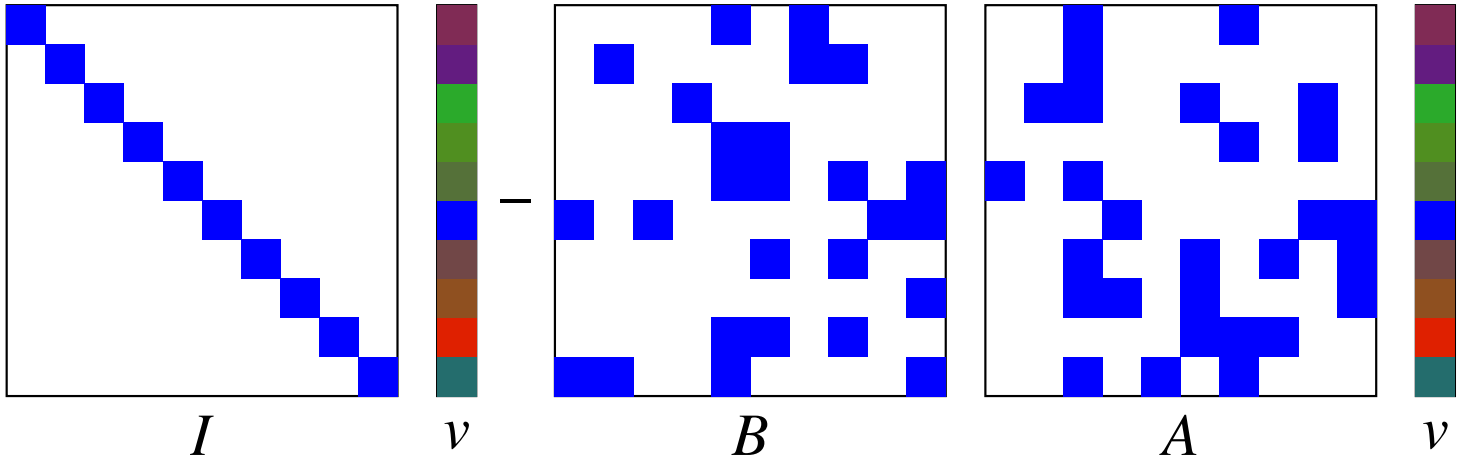




Inverse Algorithm Overview

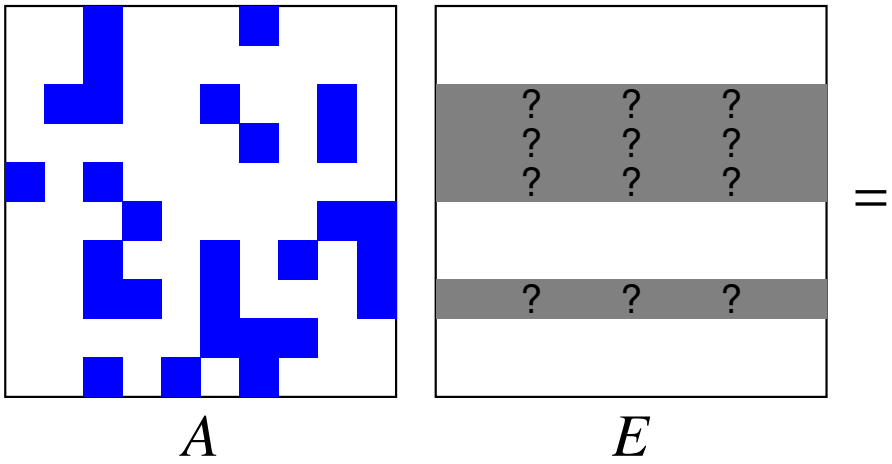
Find nonzero rows of E

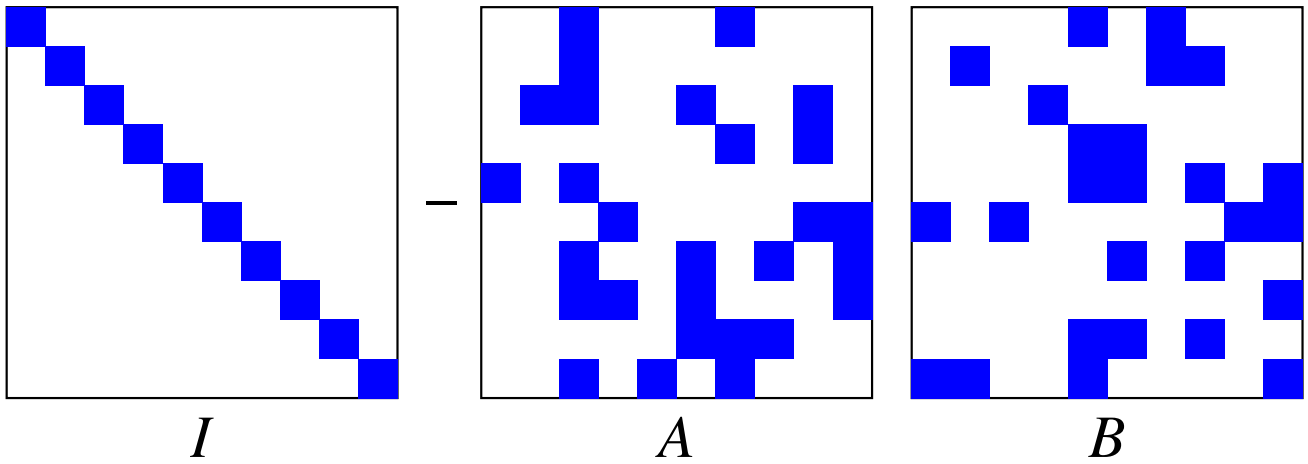




Inverse Algorithm Overview

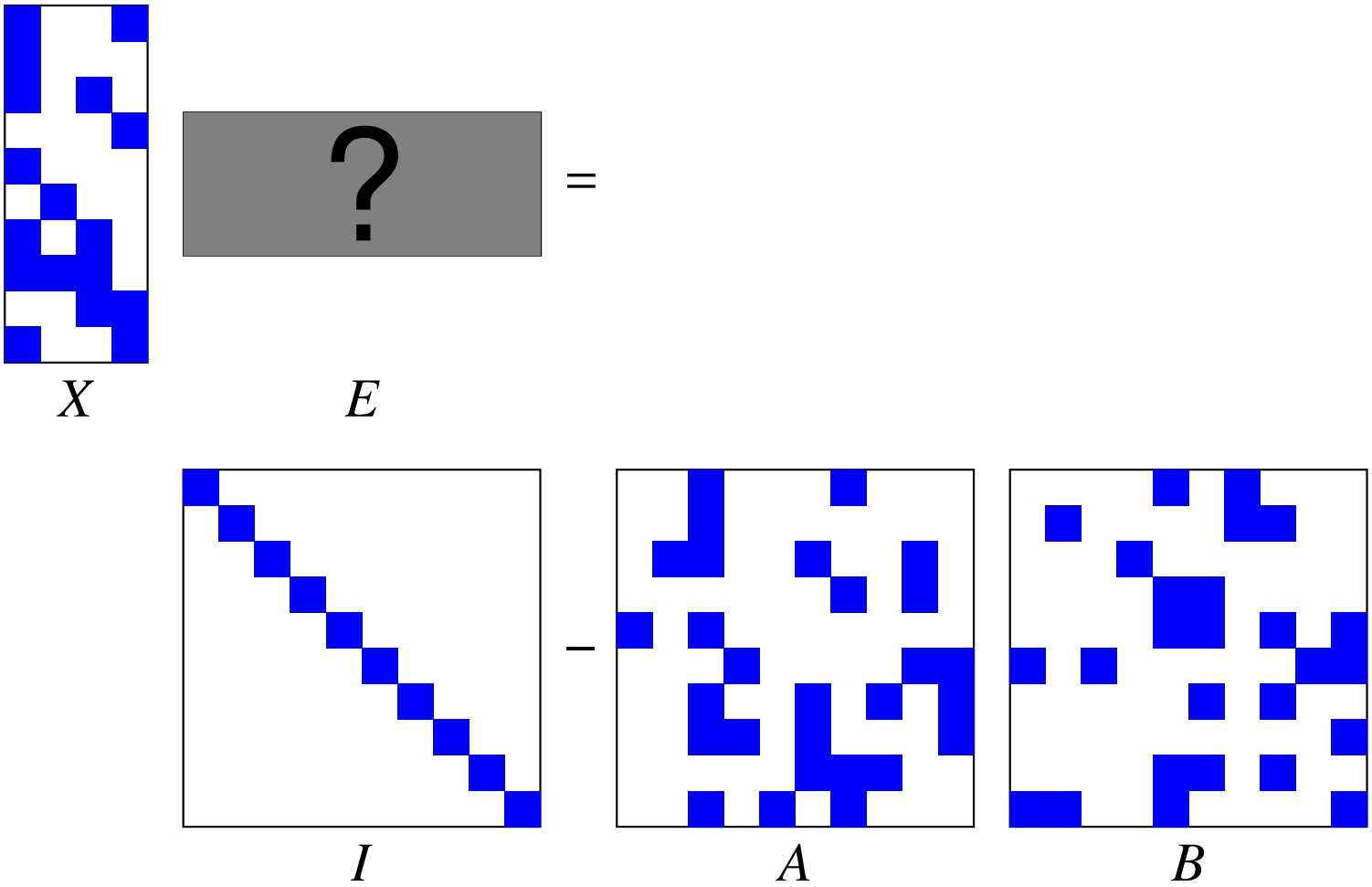
Write $AE = I - AB$





Inverse Algorithm Overview

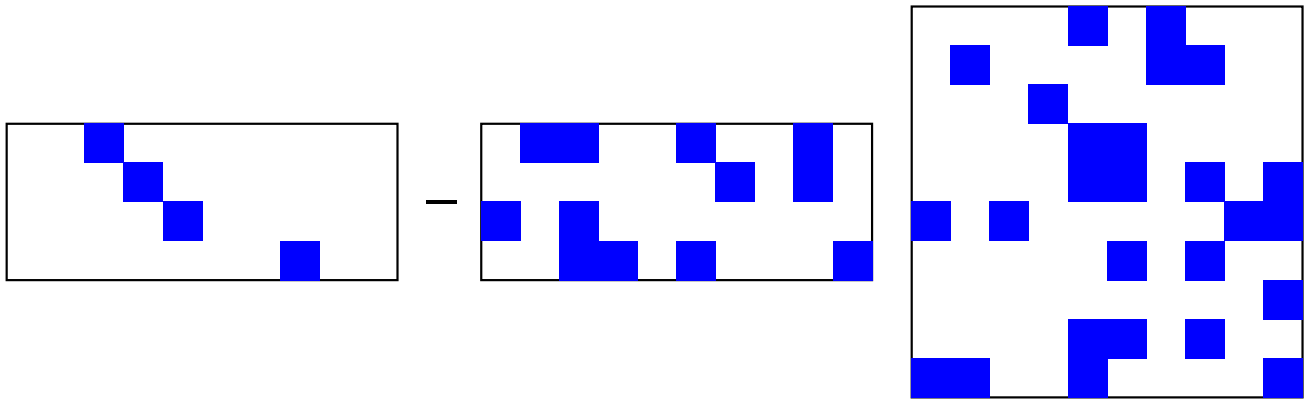
Remove zero rows of E and corresp. columns from A



Inverse Algorithm Overview

Find linear independent rows of X, remove others from both sides

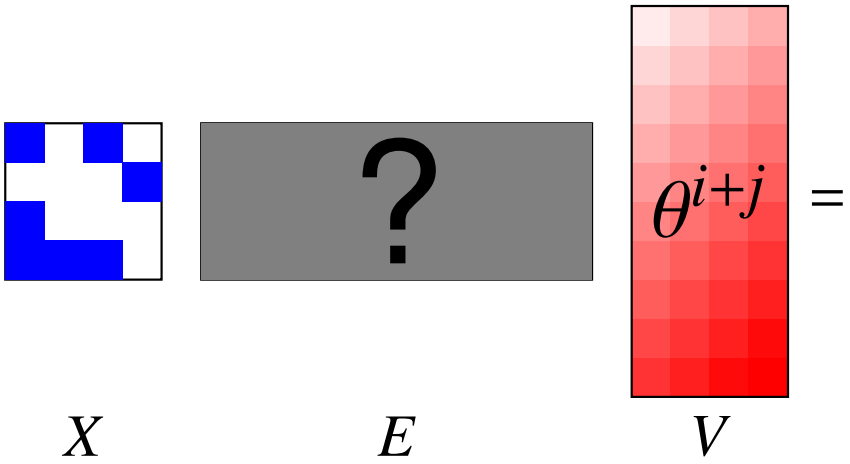
X E

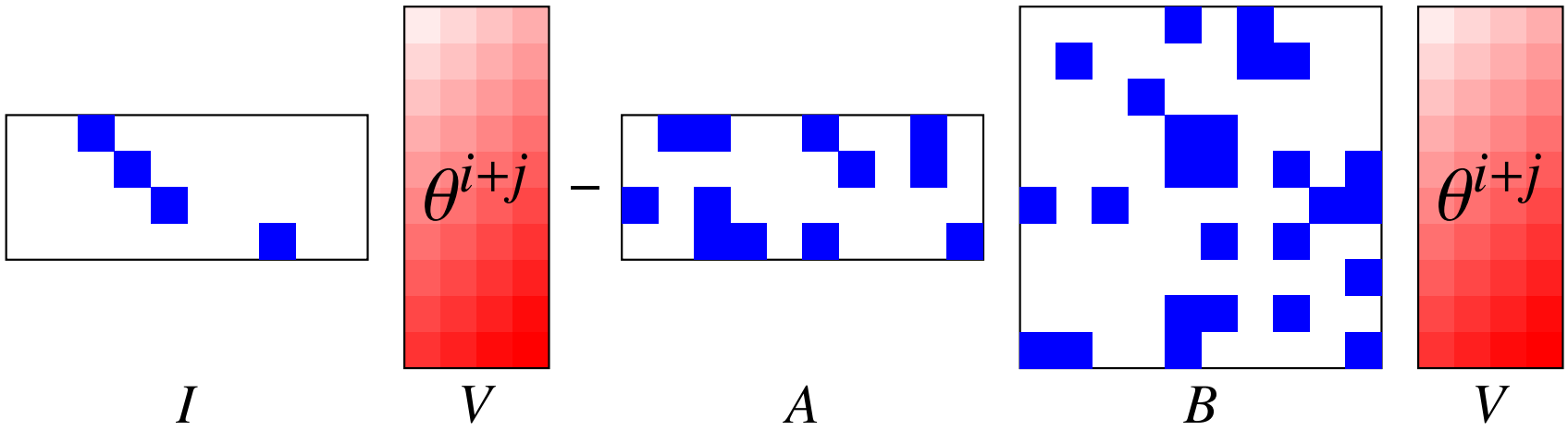


I A B

Inverse Algorithm Overview

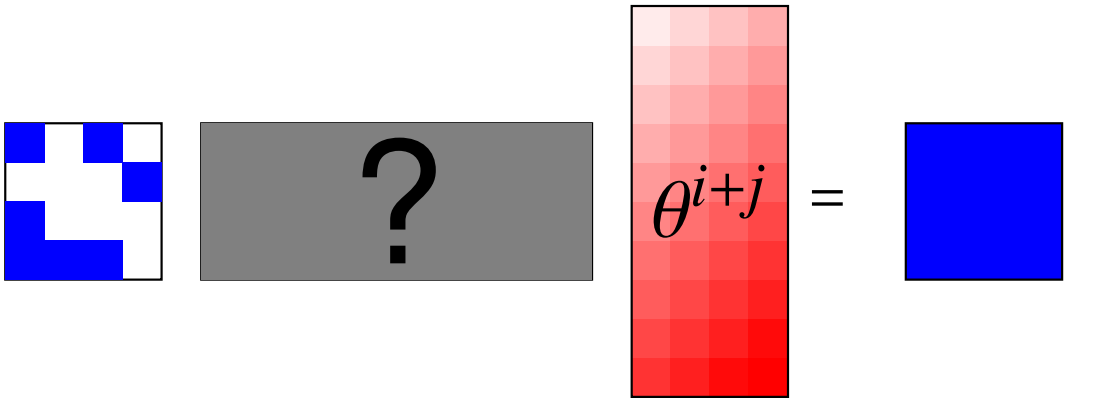
Evaluate row polynomials at first $2s$ powers of θ





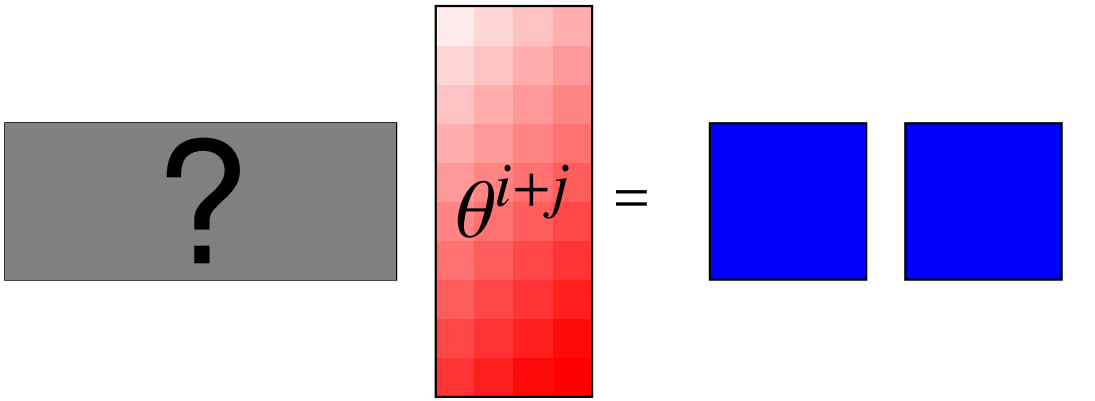
Inverse Algorithm Overview

Evaluate row polynomials at first $2s$ powers of θ



Inverse Algorithm Overview

Compute X^{-1} and apply to both sides



Inverse Complexity

Total cost in field operations is $\widetilde{O}\left(t + r^\omega + kn/\min(r, \frac{k}{r})^{3-\omega}\right)$

- n = dimension
- t = number of nonzeros in A, B, C
- k = number of errors (nonzeros in E)
- r = number of nonzero rows or columns in E, whichever is smaller
- $\omega < 2.373$, exponent of matrix multiplication

"Compact" errors	Worst case		
$r \approx \sqrt{k}$	$k \leq n^{0.73}$	$n^{0.74} \leq k \leq n$	$k \geq n$
$t + k^{0.69}n$	$t + kn$	$t + k^{2.38}$	$n^{2.38}$
$t + k^{(\omega-1)/2}n$		$t + k^\omega$	n^ω

Summary

New algorithms for:

1. **Multiplication with errors**: Computing $E = C - AB$ given A, B, C
2. **Inverse with errors**: Computing $E = A^{-1} - B$ given A, B

Both algorithms account for sparsity in the input and outputs, and can be (almost) the same cost as reading the input and output when there are few errors.

Possible application: Chinese remaindering

Error correction algorithms can be used when the output has varying-sized entries

Example

$$\begin{pmatrix} 11 & 18 & 20 & -3 \\ 1 & 14 & 9 & -23 \\ 7 & -28 & -18 & -14 \\ 28 & 22 & -16 & -30 \end{pmatrix} * \begin{pmatrix} 15 & 30 & -6 & 30 \\ -5 & 29 & 28 & 21 \\ 15 & -19 & -1 & -21 \\ -15 & -22 & -11 & -29 \end{pmatrix}$$

$$= \begin{pmatrix} 420 & 538 & 451 & 375 \\ 425 & 771 & 630 & 802 \\ 185 & 48 & -654 & 406 \\ 520 & 2442 & 794 & 2508 \end{pmatrix}$$

Possible application: Chinese remaindering

Error correction algorithms can be used when the output has varying-sized entries

Example

$$\begin{pmatrix} 11 & 18 & 20 & -3 \\ 1 & 14 & 9 & -23 \\ 7 & -28 & -18 & -14 \\ 28 & 22 & -16 & -30 \end{pmatrix} * \begin{pmatrix} 15 & 30 & -6 & 30 \\ -5 & 29 & 28 & 21 \\ 15 & -19 & -1 & -21 \\ -15 & -22 & -11 & -29 \end{pmatrix} \mod 1607$$

$$= \begin{pmatrix} 420 & 538 & 451 & 375 \\ 425 & 771 & 630 & 802 \\ 185 & 48 & -654 & 406 \\ 520 & -772 & 794 & -706 \end{pmatrix}$$

Updating with subsequent primes is an error correction problem!

Implementation

How well does this work in practice?

Preliminary implementation:

- Written in Sage
- Includes many tweaks in sparse interpolation
- Able to count field operations on actual instances

Experiments so far test the crossover point where naïve recomputation becomes better than error correction.

Experimental crossover points