An Algorithm for Inverting Matrices over GF(2) in time $O(n^3/\log n)$

Gregory Bard

University of Maryland

Applications

- Error Correcting Codes.
- Cryptography and Cryptanalysis.
- Determining a Graph Coloring.
- Systems of Polynomial Equations via the XL Algorithm.

History

- In 1970, Arlazarov, Dinic, Kronrod, and Faradzev published "On Economical Construction of the Transitive Closure of a Directed Graph."
- Finding the transitive closure of a graph is equivalent to the repeated squaring of its adjacency matrix.
- Squaring a matrix and multiplication are equivalent:

$$\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}^2 = \begin{bmatrix} A^2 & AB \\ 0 & 0 \end{bmatrix}$$

- Thus was born "The Four Russians" matrix multiplication algorithm.

Overview

- *Objective:* To find an algorithm for inverting or LUP-factoring a matrix with entries from $GF(2)$, in faster than cubic time.

Gaussian Elimination	$O(n^3)$
Method of Four Russians	$O(n^3/\log n)$
Strassen's Algorithm w/ SMIF	$O(n^{2.807})$

- Strategy: Combine both algorithms.
  - Use Strassen's Matrix Inversion Formula to cut the matrix into submatrices of size roughly 64000×64000.
  - Use the Method of Four Russians on each submatrix.
  - Glue it all back together with Strassen's Algorithm.

The Gray Code

- Discovered by Frank Gray at Bell Labs in 1953.
- Example of a Gray 3-Code:

$$(000, 001, 011, 010, 110, 111, 101, 100)$$

- Each string is different in exactly one location from its neighbors.
- All possible 3-bit strings are included.

Problem: Computing All Linear Combinations

- Suppose I had to find all $2^{10} = 1024$ vectors in the span of 10 linearly independent vectors: $v_1, \ldots, v_{10}$.
- e.g. . . . , $(v_1 + v_3 + v_7)$, $(v_3 + v_4 + v_5 + v_9)$, (0), . . .
- The naive method would require $4 \times 1024 = 4096$ vector additions, since a typical vector in the span is the sum of (on average) 5 vectors.

Solution: Computing All Linear Combinations

- But with a Gray 10-Code, only 1023 vector additions are required!

00000 00000	0
00000 00001	v_{10}
00000 00011	$v_9 + v_{10}$
00000 00010	v_9
00000 00110	$v_8 + v_9$
00000 00111	$v_8 + v_9 + v_{10}$
$\vdots$	$\vdots$
10110 00101	$v_1 + v_3 + v_4 + v_8 + v_{10}$
10110 00111	$v_1 + v_3 + v_4 + v_8 + v_9 + v_{10}$
10110 00110	$v_1 + v_3 + v_4 + v_8 + v_9$
$\vdots$	$\vdots$

Probability of Full Rank

- Suppose A is an $m \times n$ matrix filled by fair coin flips.
- What is the probability it is full rank?

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- This comes to:

$$\frac{(2^m-1)(2^m-2)\cdots(2^m-2^{i-1})\cdots(2^m-2^{n-1})}{2^{mn}} = \prod_{i=1}^{i=n} (1 - 2^{i-1-m})$$

- For large n, if A is $n \times n$ this is 29%.
- If A is $3n \times n$ this is almost 100%.

Gaussian Elimination

$$\left[\begin{array}{cccc|c|ccccc}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & \cdots & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & \cdots & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots
\end{array}\right]$$

- For each column i do
  - Find a row with 1 in col i, and swap with row i.
  - For each row $j \neq i$ do
    - If $A_{j,i} = 0$ do nothing.
    - If $A_{j,i} = 1$ do add row i to row j.

Two-at-a-time

$$\left[\begin{array}{cccc|cc|cccc}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & \cdots & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & \cdots & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots
\end{array}\right]$$

- For each column pair i and $i+1$ do
  - Do row and column swaps to arrange the above situation.
  - For each row $j \neq i, i+1$ do
    - If $(A_{j,i}, A_{j,i+1}) = (0,0)$ do nothing.
    - If $(A_{j,i}, A_{j,i+1}) = (0,1)$ add row $i+1$.
    - If $(A_{j,i}, A_{j,i+1}) = (1,0)$ add row i.
    - If $(A_{j,i}, A_{j,i+1}) = (1,1)$ add both rows.

The Algorithm

- The Method-Of-Four-Russians for Inversion consists of essentially performing k-columns of Gaussian Elimination at once, but including several tricks.
- The parameter k (an integer) will be optimized later, and usually $8 \leq k \leq 16$.
- Each iteration processes k columns, and consists of three stages.

Stage I: Create "Working Rows"

- Here one iteration has finished and we are starting iteration two. These are $k \times k$ blocks or submatrices.

- Select the next $3k$ rows, and perform Gaussian Elimination on them. This costs $\sim (4.5k^2n - 0.75k^3)$ reads/writes. Probability of failure is infinitesimal.

$$\left[\begin{array}{c|cccc} I & ? & ? & \cdots & ? \\ \hline 0 & ? & ? & \cdots & ? \\ 0 & ? & ? & \cdots & ? \\ 0 & ? & ? & \cdots & ? \\ \hline 0 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & ? & ? & \cdots & ? \end{array}\right] \Rightarrow \left[\begin{array}{c|cccc} I & ? & ? & \cdots & ? \\ \hline 0 & I & ? & \cdots & ? \\ 0 & 0 & ? & \cdots & ? \\ 0 & 0 & ? & \cdots & ? \\ \hline 0 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & ? & ? & \cdots & ? \end{array}\right]$$

Stage II: Pre-Calculate Linear Combinations

- We have k row-vectors that now are linearly independent.
- Using the Gray-code, we can cheaply calculate *all possible* linear combinations of these $2^k - 1$ non-zero vectors.
- We will use these linear-combinations to clear the k columns of all the other $m - 3k$ rows.

$$\left[\begin{array}{c|cccc} I & ? & ? & \cdots & ? \\ \hline 0 & I & ? & \cdots & ? \\ \hline 0 & 0 & ? & \cdots & ? \\ 0 & 0 & ? & \cdots & ? \\ 0 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & ? & ? & \cdots & ? \end{array}\right]$$

Stage III: Clear the Rest

- Now suppose the following two rows exist outside the $3k$ selected rows.

$$\left[\begin{array}{cccc|cccccc|cccc}
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & 0 & 0 & 1 & 1 & 0 & \cdots & 0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & 0 & 1 & 0 & 1 & 0 & \cdots & 0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
\multicolumn{4}{c}{\underbrace{\hphantom{0 \quad 0 \quad \cdots \quad 0}}_{k \text{ cols}}} & \multicolumn{6}{c}{\underbrace{\hphantom{0 \quad 1 \quad 1 \quad 0 \quad \cdots \quad 0}}_{k \text{ cols}}} & \multicolumn{4}{c}{\underbrace{\hphantom{1 \quad 1 \quad \cdots \quad 0}}_{n-2k \text{ cols}}}
\end{array}\right]$$

- The first of these needs a copy of rows 2 and 3 added to it.
- The second one needs a copy of rows 1 and 3 added to it.
- But these are already precomputed! So it's just a vector addition.

Final Complexity Analysis

- Each iteration takes care of k columns. Therefore n/k iterations are required, each of three stages:

1	Force $k \times k$ Identity	$\sim (4.5k^2n - 0.75k^3)$ reads and writes.
2	Generate Gray Code	$\sim 3(2^k(n-k))$ reads and writes.
3	Clear k Columns	$\sim 3(n-k)m$ reads and writes.

- One can show optimum occurs at $k \approx \log n$.
- Each iteration is then $\sim 6n^2$ operations, or a grand total of $\sim 6n^3/\log n = O(n^3/\log n)$.
- The LUP-factorization and minor tricks reduce 6 to 5/2.
- You can also use this algorithm for system solving, on a $n \times n + 1$ matrix, or to find pseudo-inverses of a $m \times n$ matrix.

Dim.	4,000	8,000	16,000	20,000	24,000	32,000
Gauss	19.00	138.34	1033.50	2022.29	3459.77	8061.90
k=7	7.64	–	–	–	–	–
8	7.09	51.78	–	–	–	–
9	6.90	48.83	364.74	698.67	1195.78	–
10	7.05	47.31	342.75	651.63	1107.17	2635.64
11	7.67	48.08	332.37	622.86	1051.25	2476.58
12	–	52.55	336.11	620.35	1032.38	2397.45
13	–	–	364.22	655.40	1073.45	2432.18
14	–	–	–	–	–	2657.26
Min	6.90	47.31	332.37	620.35	1032.38	2397.45
Ratio	2.75	2.97	3.11	3.26	3.35	3.36

Running Times of Gaussian Elimination vs. Method of 4 Russians (in seconds)