Efficient Problem Reductions in Linear Algebra

Erich Kaltofen
North Carolina State University
www.kaltofen.net
A simple example

\[
\begin{bmatrix}
I & A & 0 \\
0 & I & B \\
0 & 0 & I
\end{bmatrix}^{-1} = \begin{bmatrix}
I & -A & AB \\
0 & I & -B \\
0 & 0 & I
\end{bmatrix} \in \mathbb{F}^{3n \times 3n}
\]

\[\implies \text{MATMULT}_{\text{arithm.}}(n) = O(\text{MATINV}_{\text{arithm.}}(n)).\]

Further examples:

**LUDECOMP**\( (n) = O(\text{MATMULT}(n)) \) [Bunch and Hopcroft 1974]

**MATMULT**\( (n) = O(\text{DET}(n)) \) [Baur and Strassen 1983]

**CHARPOLY**\( (n) = \text{MATMULT}(n)^{1+o(1)} \) [Keller-Gehrig 1985]

**FROBFORM**\( (n) = \text{MATMULT}(n)^{1+o(1)} \) [Giesbrecht 1992]
Model: algebraic RAM over $\mathbb{F} = \mathbb{Q}(\sqrt{2})$

<table>
<thead>
<tr>
<th></th>
<th>$\sqrt{2} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$\sqrt{2} - 1$</td>
</tr>
</tbody>
</table>

Infinite input medium

| 1: READADDR | 2 |
| 2: READ     | *2 |
| 3: CONSTADDR | 1, 2 |
| 4: ADDADDR  | 1, 2 |
| 5: CONST    | *1, $\sqrt{2}$ |
| 6: DIV      | 5,*2 |
| 7: PRINT    | *1 |
| 8: HALT     | |

Fixed length program (“algorithm”)

<table>
<thead>
<tr>
<th>addr. memory</th>
<th>data memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>$\rightarrow$5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>?</td>
</tr>
</tbody>
</table>

Infinite output medium

<table>
<thead>
<tr>
<th>2 + $\sqrt{2}$</th>
<th>EOT</th>
</tr>
</thead>
</table>

Infinite addr. memory

Infinite data memory
- computes a function from $D \rightarrow E$ where $D$ is infinite
- can be programmed as a C++ template function
- defines arithmetic time and space complexities

Ambiguity through randomization: \text{RANDOM}\{\text{ADDR}\} \ i, \ j

The operand $j$ points to an address which is the cardinality of $S \subset \mathbb{F}$ from which random elements are sampled.

- Monte Carlo: “always fast, probably correct”. Examples: \texttt{isprime}

\textbf{Lemma} [DeMillo&Lipton’78, Schwartz/Zippel’79]
Let $f, g \in \mathbb{F}[x_1, \ldots, x_n], f \neq g, S \subset \mathbb{F}$.

\begin{align*}
\text{Probability}(f(a_1, \ldots, a_n) \neq g(a_1, \ldots, a_n) \mid a_i \in S) 
\geq 1 - \max\{\deg(f), \deg(g)\} / \text{cardinality}(S)
\end{align*}

sparse polynomial interpolation, factorization, minimal polynomial of a sparse matrix
Do we exactly know what the algorithm computes? E.g., in the presence of floating point arithmetic?

– Las Vegas: “always correct, probably fast”.
  Examples: polynomial factorization in $\mathbb{Z}_p[x]$, where $p \gg 2$.
  Determinant of a sparse matrix
**Theorem** [Strassen ’73; Baur and Strassen ’83; see Giesbrecht ’92] Suppose you have a Monte Carlo randomized algorithm on an algebraic random access machine that can compute the determinant of an $n \times n$ matrix in $D(n)$ arithmetic operations.

Then you have a Monte Carlo randomized algorithm on a random access machine that can multiply two $n \times n$ matrices in $O(D(n))$ arithmetic operations.

No proof is known for Las Vegas or deterministic algorithms.
Proof requires

– Eliminate superfluous tests on algebraic elements by random evaluation [DeMillo & Lipton ’78/ Schwartz/ Zippel ’79]

– \text{MATINV} \leq \text{DET}: reverse mode of automatic differentiation and [Baur & Strassen ’83]: \( (A^{-1})_{i,j} = \frac{(-1)^{i+j} \frac{\partial \det(A)}{\partial a_{j,i}}}{\det(A)} \)

– \text{C(C+I)} \leq \text{MATINV}: \textbf{C(C+I)} = (\textbf{C}^{-1} - (\textbf{C}+\textbf{I})^{-1})^{-1} \text{ and entries in } \textbf{C(C+I)} \text{ are algebraically independent [Strassen ’73]}

– Eliminate divisions from straight-line program for \textbf{C(C+I)} [Strassen/Ungar ’73]

– \text{MATMULT} \leq \text{division-free-} \textbf{C(C+I)}: \begin{bmatrix} 0 & A \\ 0 & B \end{bmatrix} \cdot \begin{bmatrix} I & A \\ 0 & I + B \end{bmatrix} = \begin{bmatrix} 0 & A + AB \\ 0 & B + B^2 \end{bmatrix}
Black box linear algebra

The black box model of a matrix

\[ y \in \mathbb{F}^n \rightarrow \begin{array}{c} \text{black box} \\ A \in \mathbb{F}^{n \times n} \text{ singular} \end{array} \rightarrow A \cdot y \in \mathbb{F}^n \]

\( \mathbb{F} \) an arbitrary, e.g., finite field

Perform linear algebra operations, e.g., \( A^{-1}b \) [Wiedemann 86] with

\[ O(n) \] black box calls and

\[ n^2(\log n)^{O(1)} \] arithmetic operations in \( \mathbb{F} \) and

\[ O(n) \] intermediate storage for field elements
**LinSolve0:** Given black box $A \in \mathbb{F}^{n \times n}$, compute $w \neq 0$ such that $Aw = 0$.

Used in sieve-based integer factoring algorithms, Las Vegas singularity and Monte-Carlo non-singularity tests.

**NonSingular ≤ LinSolve0:** For $Ax = b$ solve

$$\begin{bmatrix} A & b \\ 0^{1 \times n} & 0 \end{bmatrix} w = 0$$

and compute $x = \frac{1}{w_{n+1}} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$.

Harder (?) problem

**LinSolve1:** Given black box $A \in \mathbb{F}^{n \times n}$ (possibly singular) and $b$, compute $x$ such that $Ax = b$. 
Results from Kaltofen & Saunders 1991

Random sampling in the nullspace is equivalent to \textsc{LinSolve1}:

**random-linsolve0 \leq \textsc{linsolve1}**

select a random vector $y$ and solve $Ax = b$ for $b = Ay$ using \textsc{linsolve1}. Return $w = x - y$.

Note that $y$ is known to \textsc{linsolve1} only up to a shift by any nullspace vector.

**\textsc{linsolve1} \leq \text{random-linsolve0}**

Solve $[A \mid b] w = 0$ by random-\textsc{linsolve0}.

With probability $1 - 1/|\mathbb{F}|$ we have $w_{n+1} \neq 0$:

consider a basis $w = \sum_{i=1}^{r} c_i w^{[i]}$ and $w^{[1]}_{n+1} \neq 0$.

For any choice of $c_2, \ldots, c_r$ only one $c_1$ yields $w_{n+1} = 0$. 
Results from Kaltofen & Saunders 1991 continued

**LinSolve1 ≤ LinSolve0 + Rank**

For $r = \text{rank}(A)$ use a preconditioner

\[
\begin{bmatrix}
\tilde{A}^{[r]} & \tilde{A}^{[1,2]} \\
\tilde{A}^{[2,1]} & \tilde{A}^{[2,2]}
\end{bmatrix} = B^{[1]} \cdot A \cdot B^{[2]}
\]

such that the $r \times r$ top-left submatrix $\tilde{A}^{[r]}$ is non-singular.

Note: $B^{[i]}$ can be sparse, Toeplitz, or “butterfly” matrices [Chen et al. 2002]; we have a black box algorithm for $\tilde{A}^{[r]}$.

Solve the $r \times r$ non-singular system

\[
\tilde{A}^{[r]} \tilde{z}^{[r]} = \tilde{b}^{[r]} \quad \text{where} \quad B^{[1]} b = \begin{bmatrix}
\tilde{b}^{[r]} \\
\vdots
\end{bmatrix}
\]

and return $x = B^{[2]} \begin{bmatrix}
\tilde{z}^{[r]} \\
0
\end{bmatrix}$. 
When is $\text{PROBLEM1} \leq \text{PROBLEM2}$?

- Both $\text{PROBLEM2}$ and the black box matrix act as oracles. E.g., $\text{PROBLEM2}$ is solved for a preconditioned black box matrix.

- The algorithm for $\text{PROBLEM2}$ could be for a fixed or a generic coefficient field. E.g., $\text{PROBLEM2}$ is solved over a field extension.

- $O(1)$ versus $(\log n)^{O(1)}$ deceleration. E.g., $\text{PROBLEM2}$ is called $\log n$ times or on matrices of bigger dimensions. The black box matrix is called $O(n)$ times. 

Note: $\text{LINSOLVE1} \leq \text{PRECONDNIL}\div\text{Wiedemann/Lanczos}$
**LIN\text{SOLVE}1 \leq LIN\text{SOLVE}0**

Can assume $b = e_{n+1}$: consider

\[
\begin{bmatrix}
  A & -b \\
  0^{1 \times n} & 1 \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  1 \\
\end{bmatrix}
= 
\begin{bmatrix}
  0^n \\
  1 \\
\end{bmatrix}
\]

Why $\tilde{A} = \tilde{A} - \bar{b}c^T = [\tilde{A}_{*,1} - c_1\bar{b} | \ldots | \tilde{A}_{*,n+1} - c_{n+1}\bar{b}]$, $c_j$ random, $\tilde{A}\tilde{w} = 0$ does not yield $\bar{x} = 1/(c^T\tilde{w})\cdot\tilde{w}$ for all $\tilde{A}$:

Suppose $\tilde{A} = [0 | 0 | \tilde{A}_{*,3} | \ldots]$ and the algorithm for LIN\text{SOLVE}0 first checks if two columns are dependent. Then for any choice of $c_j$ we always get $c^T\tilde{w} = 0$.

Must hide $\bar{b}$ better: $\tilde{A} = \tilde{A}B - \bar{b}c^T$ or $\tilde{A} = (\tilde{A} - \bar{b}c^T)B$.
The curse of soft-O

\[ \log_2 n < n^{1/3 - 1/5} \quad \text{for } n \geq n_0: \quad n_0 \geq 10^{12}. \]

\[ \log_2 n < n^{1/5 - 1/7} \quad \text{for } n \geq n_0: \quad n_0 \geq 10^{37}. \]

\[ (\log_2 n)^2 < n^{1/2} \quad \text{for } n \geq n_0: \quad n_0 \geq 2^{16} = 65536. \]
**Reductions and bit complexity**

By CRA, \texttt{MATMULT} has bit complexity \(\leq n^{\omega}(\log \|AB\|)^{1+o(1)}\), where \(\omega\) is the exponent of the arithmetic complexity.

\begin{align*}
\text{DET:} & \leq n^{3.19}(\log \|A\|)^{1+o(1)} \quad [\text{Eberly, Giesbrecht, Villard 2000}] \\
& \leq n^{3.03}(\log \|A\|)^{1+o(1)} \quad [\text{Kaltofen 1995/2000}] \\
& \leq n^{2.70}(\log \|A\|)^{1+o(1)} \quad [\text{Kaltofen, Villard 2001}] \\
& \leq n^{2.38}(\log \|A\|)^{1+o(1)} \quad [\text{Storjohann 2002}] 
\end{align*}

**The argonautical quest:** How do preserve bit complexity when computing high degree objects?