... using and extending the LinBox library.

David Saunders, Bryan Youse, U Delware
Jean-Christophe Dumas, U Grenoble

Linear Algebra Module Tiny Primes

www.linalg.org
... using and extending the LinBox library.

Jean-Guillaume Dumas - U Grenoble
David Saunders, Bryan Youse - U Delaware

Integer Rank and Rank Modulo Tiny Primes
Outline

• Large matrix, small rank conjecture for Dickson SRG family

• Large matrix, large rank, resources needed

• Large rank algorithms, resources needed

• Packing schemes

• Integration of packing into LinBox

• Large rank, small rank algorithms, resources needed
Algebraic Graph Theory

Ranks and Smith forms of incidence and adjacency matrices play an important role in classification of various graphs and graph families. Andries Brouwer, … distance regular graphs: Matrix order $n \sim 10^6$. $p$-Rank to be computed. $p$, smallish $p$. Previous case done: $n = 3^{14}$, $r = 32064$, 4 days single process.

Qing Xiang, Peter Sin, David Chandler, … strongly regular graphs: Matrix order $n = 3^{16} \times 10^6$. $p$-Rank to be computed. $p > 2^{17} \sim 10^9$. Previous case done: $n \sim 10^5$, 1 file per row, amusing probs.
Dickson's Hadamard difference set

\[ G = \text{additive group of } GF(p^e) \times GF(p^e) \]

\[ D = \{ (a^2 + gb^2 \sigma, 2ab) \mid (a, b) \neq (0, 0), (a, b) \in G \} \]

where \( g \) is a generator, \( \sigma \) an automorphism of \( GF(p^e) \).

Adjacent matrix:

\[
\begin{cases}
0, & \text{otherwise} \\
1, & \text{if } e_i - e_j \in D \\
1 - d, & \text{if } e_i \neq e_j
\end{cases}
\]

Adjacency matrix: \( (G, D) \)

where \( G \) is the set of non-zero squares of a semi-field multiplication on \( GF(p^e) \).

\[ \{ (q^a, q^b) \neq (0, 0) \mid (q^a \neq q^b + q^a) \} = D \]

Difference set \( D \) of \( GF(p^e) \times GF(p^e) \) is an additive group of \( GF(p^e) \times GF(p^e) \).

Dickson's Hadamard difference set
Table 1: The Dickson SRG example computed with summation and certificate. The time units are 's' for seconds, and 'h' for hours.

<table>
<thead>
<tr>
<th>$e$</th>
<th>$n = 32e$</th>
<th>$r$ near $2^e$</th>
<th>2007t</th>
<th>2009r</th>
<th>2009t</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>43,046,721</td>
<td></td>
<td>4,782,969</td>
<td>531,441</td>
<td>65,071</td>
</tr>
<tr>
<td>7</td>
<td>32064</td>
<td></td>
<td>7283</td>
<td>1654</td>
<td>376</td>
</tr>
<tr>
<td>6</td>
<td>1.2h</td>
<td></td>
<td>80s</td>
<td>1.4s</td>
<td>0.35s</td>
</tr>
<tr>
<td>5</td>
<td>96.4h</td>
<td></td>
<td>96.4h</td>
<td>0.017h</td>
<td>0.046s</td>
</tr>
<tr>
<td>4</td>
<td>0.0012s</td>
<td></td>
<td>0.022s</td>
<td>0.95s</td>
<td>0.0012s</td>
</tr>
<tr>
<td>3</td>
<td>0.021s</td>
<td></td>
<td>0.003s</td>
<td>0.046s</td>
<td>0.0003</td>
</tr>
<tr>
<td>2</td>
<td>0.0012s</td>
<td></td>
<td>0.95s</td>
<td>0.0012s</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.021s</td>
<td></td>
<td>0.003s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conjecture: Minimal polynomial of Dickson rank sequence is $x^3 - 4x^2 - 2x + 1$.
Conjecture: Minimal polynomial of Dickson rank sequence is
\[ x^3 - 4x^2 - 2x + 1. \]
Conjecture: Minimal polynomial of Dickson rank sequence is

\[ x^3 - 4x - 2x + 1. \]

Hankel system

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
-4 & -4 & -4 & -4 & -4 \\
2 & 2 & 2 & 2 & 2 \\
-1 & -1 & -1 & -1 & -1
\end{pmatrix}
\begin{pmatrix}
376 & 1654 & 7283 & 32064 \\
85 & 376 & 1654 & 7283 \\
20 & 85 & 376 & 1654 \\
4 & 20 & 85 & 376
\end{pmatrix}
= 0
\]
It's computation is a challenge. \( r^8 \) may strengthen or disprove the conjecture.

\[
0 = \begin{pmatrix}
1 & 1 & 4 & 4 \\
1 & 1 & 4 & 4 \\
1 & 1 & 4 & 4 \\
1 & 1 & 4 & 4 \\
\end{pmatrix}
\begin{pmatrix}
8 & 16 & 32 & 64 \\
32 & 16 & 8 & 4 \\
32 & 16 & 8 & 4 \\
32 & 16 & 8 & 4 \\
\end{pmatrix}
\]

Hankel system.
Sem-normalization consists in clearing the third bit per entry.
Intermediate results carry over to the third bit (and no farther).

Normalized values are $0 = 000_2$, $1 = 001_2$, $2 = 010_2$.

Semi-normalized values are $0 = 000_2$ or $011_2$, $1 = 001_2$, $2 = 010_2$.

Thus 21 elements per 64 bit word = 2.625 elements per byte.
3-bits per field element.

3-packing
add 3-packed words

\[
( z \& \text{mask3b} ) \gg 2 + z = z
\]

\[
\wedge + x = z
\]

\[
\ldots \text{mask3b} = 0 00 100 011 011 001
\]

semi-normalized word z.

Input: packed semi-normalized words x, y.

Output: packed semi-normalized word z.
Axpy (a \times x = z, use smul and add.

packed words:

To avoid inner loop branch, apply smul at the level of vector of

\[ z = (z \land \text{mask3b}) \ll 2 \]

\[ x = z \ll 1 \]

\[
\begin{align*}
\text{case } a = 0: & \quad z = x \\
\text{case } a = 1: & \quad z = x \\
\text{case } a = 2: & \quad z = 0
\end{align*}
\]

output: \( z = a \times x \)

word x.

Input: normal field element a (e.g. 0 \cdots 010), semi-normal packed.

smul - scalar-vector multiplication
3-bitslicing

Use two bits per field element, one in each word of a 2 word pair.

\[
\begin{align*}
\cdots x_1 &= 1001, \\
\cdots x_0 &= 0111 \\
\end{align*}
\]

pair x:

- 8. elements 0, 1, 2 are represented by first three bits of the word.
- All results are normalized to these values, Boothby & Bradshaw.

Normalized values are: 0 = 002, 1 = 012, 2 = 112.

Thus 64 elements per two 64 bit words = 4 elements per byte.

(3 bits)
Semi-normalized values are 0 = 000₂ or 011₂, 1 = 001₂, 2 = 010₂.

Semi-normalization consists in clearing the third bit per entry.
Intermediate results carry over to the third bit (and no farther).

Semi-normalized values are 0 = 000₂ or 011₂, 1 = 001₂, 2 = 010₂.

* dot product: bit-wise mul, then divide and conquer (shift, add, add)

axpy: smul + add (i.e. again no special tricks)

add: 12 bit operations (6 each for z₀ and z₁).

$$z₁ = x₀ \text{ xor } x₁$$
$$z₀ = 0x₀$$

: case $a = 2$

smul:

$$\cdots \cdot \cdot 001 = x₁$$
$$\cdots \cdot \cdot 011 = x₀$$

3-bitslicing arithmetic
packing in mantissa of floats

Use arithmetic more, bit ops less. Less tight packing, less frequent normalization.

Emphasis to date is on dot (for matrix mul), Dumas, Fousse, Salvy.

For \( n \times n \) matrix and \( p = 3 \), choose such that

\[
B = 2^{p+1} < n.
\]

Key point: Highly tuned floating point matrix multiply can be used (BLAS) followed by normalization.

Then

\[
\hat{x} = z
\]

\[
\hat{y} = \hat{z} \\
\hat{z} = x
\]

\[
\begin{align*}
\exists q & \quad a_i B^q = \hat{p} \quad 0 \leq q_p \leq p - 1 \\
\end{align*}
\]

\[
\sum d_0 a_i B^q = \hat{p} \quad 0 \leq q_p \leq p - 1
\]

\[
\sum d_0 b_i B^q = \hat{h}
\]

\[
\sum d_0 a_i b_d^{-i} = \hat{z} = \hat{x}
\]
Table 2: Speed of vector and matrix operations over $\mathbb{GF}(3)$, using elements that are (a) stored as floats and using BLAS for mm, (b) stored as ints, and (c) packed.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Size</th>
<th>float</th>
<th>int</th>
<th>packed</th>
<th>pf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>15000</td>
<td>77.96</td>
<td>98.46</td>
<td>6165</td>
<td>77.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4492</td>
<td>165.9</td>
<td>21008</td>
<td>81.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120.65</td>
<td>165.9</td>
<td>4492</td>
<td>98.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix Ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>128</td>
<td>468.7</td>
<td>312.5</td>
<td>15000</td>
<td>468.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4168</td>
<td>312.5</td>
<td>4168</td>
<td>312.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Certification Theorem: Given $A$, an $n \times n$ matrix, $H$, an $n \times b$ projection matrix, and $V$, an random $n \times k$ random dense matrix, let $B = AH$ and $C = (AH | AV)$. ($B$ is $n \times b - k$, $C$ is $n \times (b + k)$).

If $r = \text{rank}(B) = \text{rank}(C)$ then $r = \text{rank}(A)$ with probability of error less than $1/q^k$, where $q$ is the cardinality of the field.

Over $\text{GF}(3)$, with $k = 13$, probability of error is less than $1$ in a million.
Corollary - 2 sided version


$y + q \times y + q// \begin{pmatrix} \Omega & \Omega \\ \Omega & \Omega \end{pmatrix} \sum = B$

$y + q \times q// \begin{pmatrix} \Lambda & \Lambda & \Lambda \\ \Lambda & \Lambda & \Lambda \\ \Lambda & \Lambda & \Lambda \end{pmatrix} \sum = \Omega B$

$$\begin{pmatrix} \Lambda & I \\ \Lambda & I \\ \Lambda & I \end{pmatrix} \begin{pmatrix} \Lambda & \Lambda & \Lambda \\ \Lambda & \Lambda & \Lambda \\ \Lambda & \Lambda & \Lambda \end{pmatrix} \begin{pmatrix} \Omega & \Omega & \Omega \\ \Omega & \Omega & \Omega \\ \Omega & \Omega & \Omega \end{pmatrix} = \begin{pmatrix} I & I & I \end{pmatrix}$$

Rows
Dickson's Hadamard difference set

\[ G = \text{additive group of } \mathbb{GF}(p^e) \times \mathbb{GF}(p^e) \]

Differenceset \( D = \{ (a^2 + gb^2 \sigma, 2ab) \mid (0, 0) \neq (a, b) \in G \} \)

where \( g \) is a generator, \( \sigma \) an automorphism of \( \mathbb{GF}(p^e) \).

Adjacent matrix:

\[
\begin{cases}
0, & \text{otherwise} \\
1, & \text{if } e - f \in D \\
1, & \text{if } f = \sigma \\\n1 - 1, & \text{if } f \\
1, & \text{if } e
\end{cases}
\]

\[ \mathcal{A} \]

Adjacency matrix:

\[ (G) \]

\( D \) is the set of non-zero squares of a semi-field multiplication on \( \mathbb{GF}(p^e)_d \).

Difference set \( D \) of \( \mathbb{GF}(p^e)_d \times \mathbb{GF}(p^e)_d \).

Difference set \( D \) of \( \mathbb{GF}(p^e)_d \times \mathbb{GF}(p^e)_d \).

Dickson's Hadamard difference set

ACGA 2009 Jun
units of scale

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Symbol</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>peta</td>
<td>P</td>
<td>15</td>
</tr>
<tr>
<td>tera</td>
<td>T</td>
<td>12</td>
</tr>
<tr>
<td>giga</td>
<td>G</td>
<td>9</td>
</tr>
<tr>
<td>mega</td>
<td>M</td>
<td>6</td>
</tr>
<tr>
<td>kilo</td>
<td>k</td>
<td>3</td>
</tr>
</tbody>
</table>

$ to retire
$ to attend ACA
$ to endow a university, cycles/sec
$ of ARRA bailout

arithmetic ops limit
Big dense matrix, small rank

Case $n = 3$ and $r \sim 2^{15}$.

Storage: $(2^{15})^2 \sim \frac{2^{30}}{2^{15}}$.

Time: $n^2 \sim 10^{18} \times 10^6 \sim 32$ years.

New algorithm uses $O(n^2 \log n)$ time. In $O(n^2)$ time it scans the big matrix once and generates a $q \times q$ matrix of the same rank, where $q$ is slightly greater than $r$. Then in $O(n^4)$ time compute the rank.

But storage of $(2^{15})^2$ (packed 2 bits/element) elements is 256 megabytes (would be 4GB at 32 bit word/element), and $(2^{15})^3$ (packed 2 bits/element) elements is 276 gigabytes.

Ideas: project to a matrix of order a little larger than $r$ while maintaining rank. Strengthened ideas: project heuristically, and certify (probabilistically).
Actual run times:

- Compute row echelon form (Part 3) - 3 hours.
- Generate block $q \times q$ matrix (Part 4) - 4 days.
Reduction phase:

\[ n^2 \text{ ops to produce block } B \text{ of order } q^{2^{17}}. \]

Rank phase:

\[ \frac{1}{8} \times 3^{33} 2^{17} = 3^{16} 2^{17}. \]

Case \[ n = 3^{16} \text{ and } \not\exists 2^{17}. \]

Dickson 3^{16}
Rank computation, current limits

Challenge: \( n = 2^{20} \), time 270', memory 240.

Small challenge: \( n = 2^{17} \), time 251', memory 234.

Routine: \( n = 2^{15} \), time 249', memory 230.

\( u \sim \frac{n}{3} \) space. Limit is \( n/3 \) if \( r \sim n \).

Large matrix, large rank (Brouwer's problem)

Challenge: \( n = 2^{25} \), time 271', memory 250 + 251', memory 234. 222, 215.

\( u = 2^{30} \).

Routine (with packing): \( n = 2^{19} \), time 215', memory 238 + 249', memory 230.

Large matrix, small rank Dickson problem:

**Rank computation, current limits**
from Field to...

typed Field

code using rank

//

template<class Field>
{

   //elsewhere defined

   //elsewhere defined

   //matrix representations.

   Field Types are template parameters to generic solutions and to

   Field Types, and their properties.

   FieldTypes and interface/archetype which specifies the member functions,

   concept/interface/archetype which specifies the member functions,

   In linbox the Field F is an object of a type FIELD meeting a Field

   ...

   ...

   from Field to

   ...
\text{Field}(\mathbb{F}(x,\mathbb{F}(x',\mathbb{F}(x',\mathbb{F}(x')))))

\text{Field}(\mathbb{F}(x'))

\text{DenseMatrix}(\mathbb{F}, n, \mathbb{F})

\text{DenseMatrix}(\mathbb{F}, n, \mathbb{F})

\text{DenseMatrix}(\mathbb{F}, n, \mathbb{F})

\text{DenseMatrix}(\mathbb{F}, n, \mathbb{F})
class MatrixDomain {
    // BLAS-like functionality
    // may encapsulate packing, delayed normalizations, parallelism
    class Scalar;
    class Block;
    // Block::Entry may be packed word, may be unnormalized.
    class Matrix;
    // Matrix::Entry may be packed word, may be unnormalized.
    template<class MatrixDomain> DenseMatrix;
    template<class MatrixDomain> void rank(int r, const MatrixDomain::Matrix& A);
}

...