Large Matrix, small Rank

David Saunders & Bryan Youse

U of Delaware

... with implementations in the LinBox library.

www.linalg.org

Outline

- motivation in algebraic graph theory, Paley-type graphs
- rank conjecture for Dickson SRG family.
- rank algorithm for when rank ¡¡ matrix order
- heuristic plus certificate
- computational details
- implications for exact linear algebra software

Difference sets, Paley type graphs

Partial difference sets are in 1-1 correspondence with the strongly regular graphs having a regular automorphism.

Paley graph:

vertices: elements of a field GF(p^e).

edges: have edge (a, b) if $a - b$ is a square.

Conjecture: Paley graph adjacency matrix has lowest p-rank of all SRG. (falsified by Qing Xiang)
...promotes interest in p-ranks of families of SRG. How low can these ranks get?

Paley type graph: other difference sets

For new difference sets, replace field by semi-field (non-associative).

Families of semi-field defined SRG: Pseudo-Paley, Cohen-Ganley, Dickson, etc.

Dickson's Hadamard difference set

G = additive group of $\mathrm{GF}(p^e) \times G\mathrm{F}(p^e)$

Difference set $\mathrm{D} = \{(a^2 + g\sigma(b^2), 2ab) \mid 0 \neq (a,b) \in \mathrm{G}\}$

where g is a generator, σ an automorphism of $\mathrm{GF}(p^e)$.

D is the set of non-zero squares of a semi-field multiplication on G.

Adjacency matrix $A = (a_{i,j})$, is $n \times n$, where $n = p^{2e}$ and

$$a_{i,j} = \begin{cases} p-1, & \text{if } i = j, \\ 1, & \text{if } e_i - e_j \in D, \\ 0, & \text{otherwise.} \end{cases}$$

For p = 3,

exponent	order	rank
e	$n = 3^{2e}$	r near 2^{2e}
1	9	4
2	81	20
3	729	85
4	6561	376
5	59049	1654
6	531441	7283
7	4782969	32064
8	43046721	?

Hankel system

Is this sequence linear recurrent?

4, 20, 85, 367, 1654, 7283, ...

$$\begin{pmatrix} 4 & 20 & 85 \\ 20 & 85 & 376 \\ 85 & 376 & 1654 \\ & & \\ & & \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 376 \\ 1654 \\ 7283 \\ \\ \end{pmatrix}$$

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 \\ 20 & 85 & 376 \\ 85 & 376 & 1654 \\ 376 & 1654 & 7283 \\ \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 376 \\ 1654 \\ 7283 \\ 32064 \\ \end{pmatrix}$$

Conjecture: Minimal polynomial of Dickson rank sequence is $x^3 - 4x - 2x + 1$.

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 \\ 20 & 85 & 376 \\ 85 & 376 & 1654 \\ 376 & 1654 & 7283 \\ 1654 & 7283 & 32064 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 376 \\ 1654 \\ 7283 \\ 32064 \\ r_8 \end{pmatrix}$$

r_8 may disprove or strengthen the conjecture.

It's computation is a challenge.

Algorithm template: Matrix Rank

Given $n \times n$ matrix A of rank $r << n$, compute r.

1. Choose block size b so that $r < b << n$.

2. Choose u $(b \times n)$ and v $(n \times b)$ and compute

$$d = uAv \quad (d \text{ is } b \times b)$$

3. Compute $r = \text{rank}(d)$.

4. If r is sufficiently less than b, return r as the rank of A, otherwise "Block size b is too small".

Think of projectors as strips of blocks.

We want u and v composed of easy to store and easy to use $b \times b$ blocks.

For example, block size $b, n = 3b$, overall dimensions are $(b \times n)(n \times n)(n \times b)$.

$$d = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \times \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

i.e.$d = \sum_i \sum_j u_i A_{i,j} v_j = \sum_i u_i \sum_j A_{i,j} v_j$

Step 2, what projectors u, v?

$(b \times b) = (b \times n)(n \times n)(n \times b)$

1. u and v random: if $b - \text{rank}(d) > 20$, probability that $\text{rank}(d) \neq \text{rank}(A)$ is less than $1/10^6$. [Cooperman and Havas]

2. u_i and v_j butterfly matrices: probability

   of success is high if field cardinality is high enough. [this paper (theory)]

3. $$u_i = \begin{bmatrix} I \\ r_u \end{bmatrix} \text{ and} v_j = \begin{bmatrix} I & r_v \end{bmatrix},$$

   where r_u consists of 20 random rows and r_v consists of 20 random cols. [this paper (practice)]

Idea: With care in choice of u_i, v_j, the probability that $\mathrm{rank}(d) = \mathrm{rank}(A)$ is high. Note that always $\mathrm{rank}(d) \leq \mathrm{rank}(A)$.

Why proceed this way? Ans: time and memory, especially memory.

For $n = 3^{14}$, we needed $b = 2^{15}$ and blocking factor $b/n = 146$ (storage circa 1G entries, number of blocks $\sim$ 21 thousand).

Step 1, what block size b?

If b is too small, we will learn this from the algorithm.

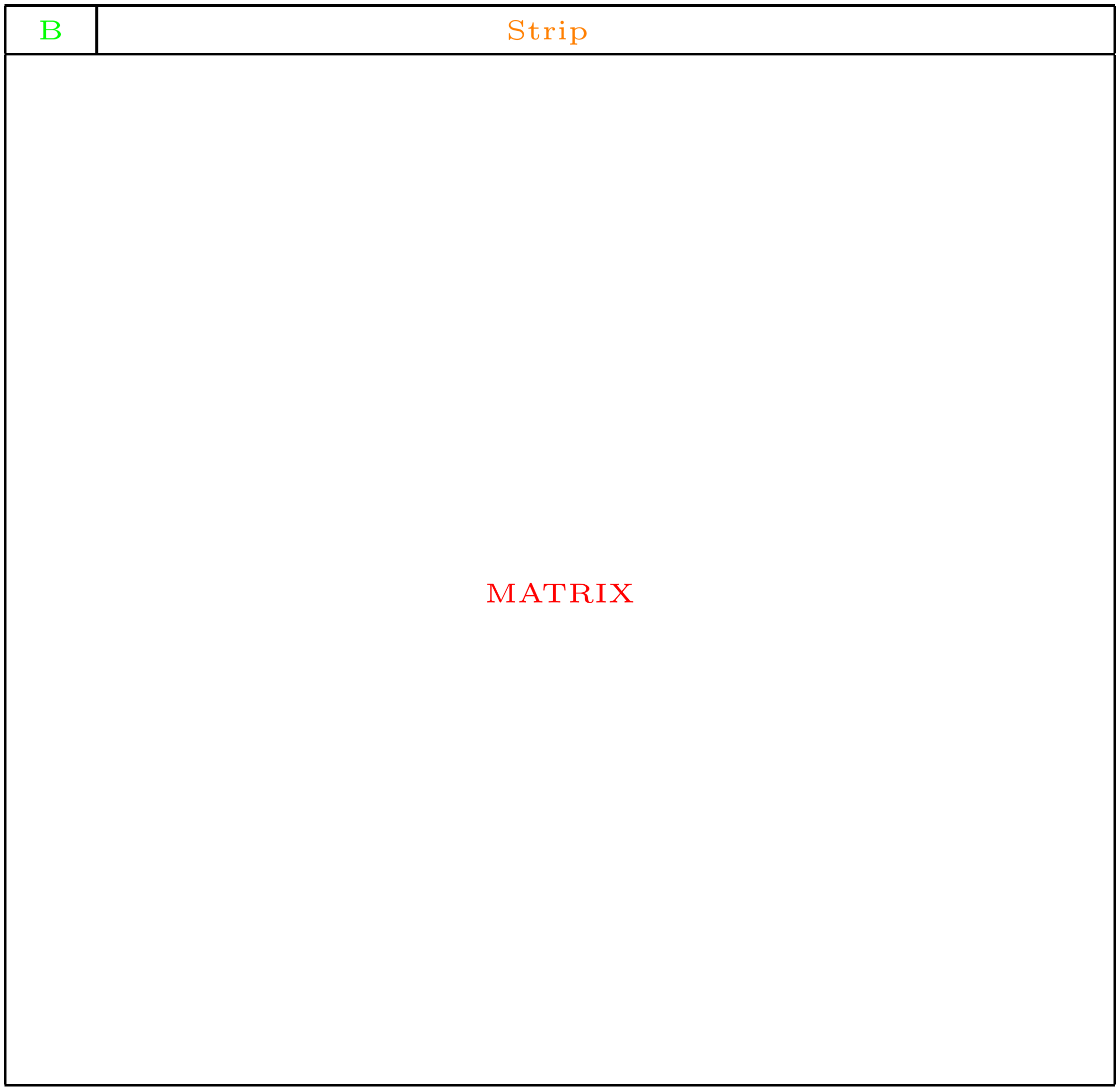
IF b is too large, we waste time and memory.

The application may suggest a value for b (as happens for the Dickson graphs).

$b = n^{2/3}$ could be used.

Recursive doubling could be used (at cost of a log(b) factor in the runtime)

Henceforward we assume b is determined.


B
Strip
MATRIX

For $n = 3^{16}$, block size is 2^{17}, blocking factor is 328 (storage circa 16G, number of blocks $\sim$ 100 thousand).

Block memory

$$b * b = 2^{34}$$

Strip memory

$$b * n > 2^{42}$$

Matrix memory

$$n * n \sim 2^{51}$$

$$d = uAv, \quad \text{goal: } \mathrm{rank}(d) = \mathrm{rank}(A).$$

Butterfly preconditioner/projector

Butterfly stage 1 of $\lg(b)$ stages, $b = 8$

$$\begin{bmatrix} a & b & & & & & & \\ c & d & & & & & & \\ & & * & * & & & & \\ & & * & * & & & & \\ & & & & * & * & & \\ & & & & * & * & & \\ & & & & & & * & * \\ & & & & & & * & * \end{bmatrix}$$

Butterfly stage 2

$$\begin{bmatrix} a & & b & & & & & \\ & * & & * & & & & \\ c & & d & & & & & \\ & * & & * & & & & \\ & & & & * & & * & \\ & & & & & * & & * \\ & & & & * & & * & \\ & & & & & * & & * \end{bmatrix}$$

Butterfly stage 3

$$\begin{bmatrix} a & & & & b & & & \\ & * & & & & * & & \\ & & * & & & & * & \\ & & & * & & & & * \\ c & & & & d & & & \\ & * & & & & * & & \\ & & * & & & & * & \\ & & & * & & & & * \end{bmatrix}$$

Butterfly preconditioner, $M \longrightarrow MB$

Multiply M by butterfly B costs O(b^2log$b$) ops, and storage of B is O($b$log(b)).

Butterfly parameters can be set to permute k arbitrary columns into the first k positions.

Butterfly parameters can be set to permute k arbitrary rows into positions $i, i+1, \ldots, k-1$. (i.e. into any contiguous chunk)

Therefore: $n \times b$ strip matrix v of $b \times b$ butterflies, $v = (B_1, \ldots, B_{n/b})^T$, can permute r columns (of n in A) into first r positions (of b in Av).

$$\left(\begin{array}{cccccc|cccccc|cccccc} 0 & * & 0 & * & 0 & 0 & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & * & 0 & * & 0 & 0 & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * & 0 & 0 \end{array}\right)$$

$$\left(\begin{array}{cccccc|cccccc|cccccc} * & * & 0 & 0 & 0 & 0 & 0 & 0 & * & * & 0 & 0 & 0 & 0 & 0 & 0 & * & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & 0 & 0 & 0 & 0 & 0 & 0 & * & * & 0 & 0 & 0 & 0 & 0 & 0 & * & 0 \end{array}\right)$$

$$\left(\begin{array}{cccccc} * & * & * & * & * & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & * & * & 0 \end{array}\right)$$

Theorem. Using preconditioner/projectors of random butterfly blocks yields an algorith that gives correct rank with high probability when field cardinality is large enough (for the random choices in butterflies) and uses $\mathrm{O}(n^2 \log(b))$ field ops and $\mathrm{O}(b^2 + n\log(b))$ memory.

Corrolary. The rank computation is in $\mathrm{O}(r^3)$, so overall runtime is $O^{\sim}(n^2 + r^3)$, which is optimal when $r < n^{2/3}$.

Hueristic plus certificate approach

Idea: (a) Partition A into $b \times b$ blocks. Add up all the blocks. Hope the rank of this sum is the rank of A. (b) Probabilistically certify by augmenting with a few random linear combinations of the rows and columns of A.

Thus, $d = uAv$, with

$$u_i = \begin{bmatrix} I \\ r_u \end{bmatrix} \text{and} v_j = \begin{bmatrix} I & r_v \end{bmatrix},$$

where r_u consists of 20 random rows and r_v consists of 20 random cols.

probabilistic certification

(One sided) Certification Theorem: Given A, an $n \times n$ matrix,
U, an $n \times b$ projection matrix, and
V, an random $n \times k$ random dense matrix,

let $B = AU$ and $C = (AU|AV)$.
(B is $n \times b$, C is $n \times (b+k)$.)

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

Remark: We can compute $\text{rank}(AU)$ and $\text{rank}(AU|AV)$ in one elimination.

Dickson SRG					
e	$n = 3^{2e}$	r near 2^{2e}	2007t	2009r	2009t
1	9	4	-	-	-
2	81	20	0.021s	0.0003	0.0012s
3	729	85	0.35s	0.003s	0.022s
4	6,561	376	33.3s	0.046s	0.95s
5	59,049	1654	1800s	1.4s	61s
6	531,441	7283	46.7h	0.02h	1.2h
7	4,782,969	32064	-	1.2h	96.4h
8	43,046,721	?	-	-	-

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

3-packing

Three bits per field element.

Thus 21 elements per 64 bit word = 2.625 elements per byte.
(unpacked - int or float - 0.25 elements per byte)

(eg. 0 010 ... 000 001 010 011)

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

Semi-normalized values are $0 = 000_2$ or $011_2, 1 = 001_2, 2 = 010_2$.
Intermediate results carry over to the third bit (and no farther).

Semi-normalization consists in clearing the third bit per entry.

add 3-packed words

input: packed semi-normalized words x, y. output: packed semi-normalized word z.

```
mask3b = 0 001 001 001 ...
z = x + y
z = z + (( z & mask3b ) >> 2 )
```

Scalar mul and axpy (mul-add): only significant case is when scalar is 2. Avoid inner loop branch by operating at vector level.

3-bitslicing

Use two bits per field element, one in each word of a 2 word pair (in corresponding bit positions).

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

Normalized values are $0 = 00_2, 1 = 01_2, 2 = 11_2$.
(all results are normalized to these values), Boothby & Bradshaw.

eg. elements 0,1,2 are represented by first three bits of the word pair x:

```
x0 = 011....
x1 = 001....
```

3-bitslicing arithmetic

```
    x0 = 011....
    x1 = 001....
```

smul:

```
case a = 2:
    z0 = x0
    z1 = x0 xor x1
```

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

axpy: smul + add

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 d such that $B = 2^{d+1} > 4n$.

$x = \sum_0^d a_i B^i$

$y = \sum_0^d b_i B^i$

$z = xy$

Then $z_d = \sum_0^d a_i b_{d-i}$.

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

GF(3) Arithmetic Comparison (MegaFFops)					
Operation	Size	float	int	sliced	pf
Vector Ops					
add	10^7	120.65	165.9	4492	
scalar mul	10^7	81.15	136.5	21008	
axpy	10^7	77.96	98.46	6165	
Matrix Ops					
mv	15000	468.7	312.5	4168	
mm	10^3	3835	350.9	3226	20k

Table 2: Speed of vector and matrix operations over GF(3), using elements that are a) stored as floats and using BLAS for mm, b) stored as ints, c) bit sliced, and d) compressed floats.

Conclusion

- Butterfly projection provides an optimal Monte Carlo rank algorithm for matrices over a large field when the rank is small.
- Block-sum-heuristic-with-Monte-Carlo-certificate algorithm plus bit-slicing enabled us to compute the 3-rank of the 7th Dickson matrix, an approximately 5 million $\times$ 5 million dense matrix, and formulate a conjecture about the Dickson semi-field difference sets.
- In addition, parallelism will be needed (and will be sufficient) to allow us to compute the rank of the 8th Dickson matrix of order 3^{16} (has 2^{51} entries).
- Effective delivery of packing, of semi-normalized values (and of parallelism) to the generic algorithms and users causes software design issues. LinBox can and must take a design evolution step to address this.