An Implementation of Orrick's Algorithm

Paul Zimmermann
(with J.-a. Osborn, R. P. Brent and W. Orrick)

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND-EST

ANU
May 14, 2010

Supported by ANU and ANU-INRIA associate team
Algorithms, Numbers, Computers

Reference: *The maximal* $\{-1, 1\}$*-determinant of order* 15, Will Orrick, Metrika (2005), extended version on arXiv:math/0401179v1.

Brute force approach:

- Stage 1: Find a candidate Gram matrix M (this talk).
- Stage 2: Try to decompose M into RR^T or R^TR (RPB's talk).

Candidate Gram matrix

1. must be symmetric and positive definite
2. $M_{i,i} = n$
3. $M_{i,j} \equiv n \bmod 4$ for $i \neq j$, $|M_{i,j}| \leq n - 2$
4. $\det(M) = d^2$
5. $d \geq d_{\min}$

Remark: since d has a factor 2^{n-1} we often omit this factor.

Orrick's Algorithm (sketch)

1. start from the sub-matrix $M = (n)$
2. at each step, for each possible matrix M_{r-1} of order $r-1$, and each admissible vector f, construct the matrix

$$M_r = \begin{pmatrix} M_{r-1} & f \\ f^T & n \end{pmatrix}$$

3. if $r = n$, check $\det M_r = d^2 \geq d_{\min}^2$ and check M_r is lexicographically maximal. If yes print the candidate matrix.
4. if $r < n$, evaluate

$$d = \begin{vmatrix} M_r & \gamma \\ \gamma^T & 1 \end{vmatrix}$$

for each possible vector γ, until we find a large enough d. If no good d is found, M_r is discarded. Otherwise continue.

Lexicographically maximal matrix

Lexicographically maximal matrix:

$$\begin{pmatrix} 11 & 3 & 3 \\ 3 & 11 & -1 \\ 3 & -1 & 11 \end{pmatrix}$$

Equivalent non-lexicographically maximal matrix:

$$\begin{pmatrix} 11 & 3 & -1 \\ 3 & 11 & 3 \\ -1 & 3 & 11 \end{pmatrix}$$

Discarding non-lexicographically maximal sub-matrices is crucial (especially at small depth $r < n$).
But it might be expensive: up to $n!$ permutations to try!

Block structure

$$\begin{pmatrix} 11 & 3 & 3 & -1 & -1 \\ 3 & 11 & -1 & -1 & -1 \\ 3 & -1 & 11 & -1 & -1 \\ -1 & -1 & -1 & 11 & 3 \\ -1 & -1 & -1 & 3 & 11 \end{pmatrix}$$

$$\begin{pmatrix} 11 & 3 & 3 & & \\ 3 & 11 & -1 & & \\ 3 & -1 & 11 & & \\ & & & 11 & 3 \\ & & & 3 & 11 \end{pmatrix}$$

In a given block, the largest (absolute) non-diagonal element must appear in $(2, 1)$ position:

$$\begin{pmatrix} 11 & -9 & 3 & & \\ -9 & 11 & -1 & & \\ 3 & -1 & 11 & & \\ & & & 11 & 7 \\ & & & 7 & 11 \end{pmatrix}$$

Also, that largest non-diagonal element cannot increase from one block to the next one.

The following matrix is lexicographically maximal:

$$\begin{pmatrix} 11 & -9 & & & \\ -9 & 11 & & & \\ & & 11 & 7 & 3 \\ & & 7 & 11 & -1 \\ & & 3 & -1 & 11 \end{pmatrix}$$

The following is not:

$$\begin{pmatrix} 11 & -9 & & & \\ -9 & 11 & & & \\ & & 11 & -9 & 3 \\ & & -9 & 11 & -1 \\ & & 3 & -1 & 11 \end{pmatrix}$$

Step 4 implements the theorem of Moyssiadis and Kounias:

Theorem

Let $M = \begin{pmatrix} D_r & B \\ B^T & A \end{pmatrix}$ *be a symmetric, positive definite matrix of order* m*, with* $|M_{i,j}| \geq c$*, where* D_r *is a square matrix of order* r*,* $A_{i,i} = n$*, and the columns of* B *are taken from some set* Γ*, with* $|B_{i,j}| \geq c$*. Then*

$$\det M \leq (n-c)^{m-r-1}[(n-c)\det D_r + (m-r)\max(0,d)].$$

Vectors f in step 2

Due to the lexicographic condition, an appended vector f cannot have an element larger than the previous one:

$$\begin{pmatrix} 11 & 7 & -9 \\ 7 & 11 & -1 \\ -9 & -1 & 11 \end{pmatrix}$$

If f has only minimal elements, we start a new block:

$$\begin{pmatrix} 11 & 7 & -1 \\ 7 & 11 & -1 \\ -1 & -1 & 11 \end{pmatrix}$$

Our C program `algo2.c` is about 2000 lines long:

- rank-1 updates for the determinants and inverse matrices
- uses the GMP library to avoid rounding errors in determinants and inverse matrices (except in Step 4, but uses a priori rigorous bound of the rounding error)
- uses integer arithmetic in the bound computation to avoid rounding errors
- compute equivalent classes where two rows/columns are equivalent if permuting them does not change the (sub)matrix
- for the `IsLexMax` test, first try a transposition between two rows/columns, then uses equivalent classes, and try up to 50000 permutations
- we did not implement a full `IsLexMax` test, thus the program may return two equivalent matrices (but does not miss matrices, up to bugs)
- implements Will's new bound for $n = 3 \bmod 4$

$n = 15$

```
patate% time ./algo2 15 418037760
...
#1: det^(1/2)/2^14=26244 641ms:
#2: det^(1/2)/2^14=26244 641ms:
#3: det^(1/2)/2^14=25515 6143ms:
#4: det^(1/2)/2^14=25515 9963ms:
...
# Total cputime = 25533ms (det=15523ms (dbl=9066ms), IsLexMax=4289ms)
```

```
#1: det^(1/2)/2^14=26244 641ms:
[15, 3, 3,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[ 3,15, 3,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[ 3, 3,15,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,15, 3, 3,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1, 3,15, 3,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1, 3, 3,15,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,15, 3, 3,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1, 3,15, 3,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1, 3, 3,15,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,15, 3, 3,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1, 3,15, 3,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1, 3, 3,15,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,15, 3,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 3,15,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,15]
```

$$
\begin{pmatrix}
15 & 3 & 3 & & & & & & & & & & & & \\
3 & 15 & 3 & & & & & & & & & & & & \\
3 & 3 & 15 & & & & & & & & & & & & \\
 & & & 15 & 3 & 3 & & & & & & & & & \\
 & & & 3 & 15 & 3 & & & & & & & & & \\
 & & & 3 & 3 & 15 & & & & & & & & & \\
 & & & & & & 15 & 3 & 3 & & & & & & \\
 & & & & & & 3 & 15 & 3 & & & & & & \\
 & & & & & & 3 & 3 & 15 & & & & & & \\
 & & & & & & & & & 15 & 3 & 3 & & & \\
 & & & & & & & & & 3 & 15 & 3 & & & \\
 & & & & & & & & & 3 & 3 & 15 & & & \\
 & & & & & & & & & & & & 15 & 3 & \\
 & & & & & & & & & & & & 3 & 15 & \\
 & & & & & & & & & & & & & & 15
\end{pmatrix}
$$

For $n = 15$, $d_{\min} = 25515 = 105 \cdot 3^5$, our C program finds 4 candidate matrices in about 26 seconds.

In 2004, using Mathematica, Will Orrick needed 7 hours.

The speedup is about 1000.

$n = 15$: statistics

```
# Det. tests 2233, square tests = 4/199 (2.01%)
# Step 4 calls 592496, aver. len 84.19, aver. comp. 76.12
# r=2: calls 7, failures 3, aver. len 49.00, aver. comp. 21.57
# r=3: calls 30, failures 8, aver. len 43.33, aver. comp. 16.57
# r=4: calls 228, failures 109, aver. len 85.52, aver. comp. 53.18
# r=5: calls 1603, failures 994, aver. len 110.56, aver. comp. 79.91
# r=6: calls 8068, failures 5419, aver. len 127.02, aver. comp. 92.63
# r=7: calls 39564, failures 30371, aver. len 149.42, aver. comp. 121.34
# r=8: calls 131475, failures 112224, aver. len 154.11, aver. comp. 139.2
# r=9: calls 188712, failures 165758, aver. len 84.15, aver. comp. 78.94
# r=10: calls 133090, failures 116720, aver. len 39.49, aver. comp. 37.44
# r=11: calls 58064, failures 51094, aver. len 18.13, aver. comp. 16.85
# r=12: calls 18042, failures 13649, aver. len 11.53, aver. comp. 10.18
# r=13: calls 7779, failures 4057, aver. len 7.22, aver. comp. 5.08
# r=14: calls 5834, failures 4282, aver. len 5.76, aver. comp. 4.70
# Total number of traversed nodes: Step 2=63249, Step 4=592496
# det(M)>0 in Step 4: 592496 nodes
# old test passed in Step 4: 87808 nodes
# Will's improved bound excluded 3032/3076 nodes
```

$n = 19$

Ehlich's bound is $\approx 854 \cdot 4^6$.

Tried $833 \cdot 4^6$, found 9 candidate Gram matrices (computation done by Richard using 50 parallel jobs in about 900 hours).

More results in RPB's talk.

$n = 23$

Ehlich's bound is $\approx 45506 \cdot 5^6$.

Partial search on $45000 \cdot 5^6$, found no matrix so far.

Richard started a parallel search on $42411 \cdot 5^6$ (where one decomposable matrix is known) found 278 matrices after 130 hours.

$n = 27$

Ehlich's bound is $\approx 564 \cdot 6^{11}$.

Partial search on $560 \cdot 6^{11}$, found no matrix so far.

$546 \cdot 6^{11}$: one decomposable matrix known.

$n = 29$

Complete search on $330 \cdot 7^{12}$: RPB found 5962 matrices in 542 hours (wall clock time).

$329 \cdot 7^{12}$: RPB found 9587 matrices in 2800 hours (incomplete search).

$320 \cdot 7^{12}$: one decomposable matrix known.

$n = 33$

Complete search on $471 \cdot 8^{14}$: RPB found 9054 matrices.

$464 \cdot 8^{14}$: RPB found 86279 matrices (incomplete search, estimated 50% done).

$441 \cdot 8^{14}$: one decomposable matrix known.

$n = 37$

Complete search on $648 \cdot 9^{16}$: found 807 matrices in 78 hours.

More results in RPB's talk.

$n = 45$

Complete search on $99 \cdot 11^{21}$: found 1495 matrices in 335 hours.

$89 \cdot 11^{21}$: new record from Will (August 2009).

$83 \cdot 11^{21}$: old record.

$n = 49$

Complete search on $114 \cdot 12^{23}$: found 168 matrices.

$96 \cdot 12^{23}$: current record.

$n = 53$

Complete search on $129 \cdot 13^{25}$: found 220 matrices.

$105 \cdot 13^{25}$: current record.

$n = 57$

Complete search on $145 \cdot 14^{27}$: found 128 matrices.

$142 \cdot 14^{27}$: conjectured maxdet.

$133 \cdot 14^{27}$: current record.

How to speed-up the search?

- run the search in parallel: already tried (RPB). We could however implement a checkpoint mechanism.
- if only one solution is enough, we could try a randomized search (works well for the decomposition). At each node of the search tree, go down in a random branch. At the bottom of the tree, do backtracking.
- search only for matrices with bounded block size (already implemented).

Bounded block-size for $n = 15$

Using $d_{\min} = 105 \cdot 3^5$:

No bound: 4 matrices in 25.5s.
Bound 3: 2 matrices in 0.018s (both $108 \cdot 3^5$).
Bound 4: 3 matrices in 0.156s.
Bound 5: 3 matrices in 0.799s.
Bound 6: 4 matrices in 3.183s.

Bounded block-size for $n = 19$

Using $d_{\min} = 833 \cdot 4^6$:

No bound: 9 matrices in 188 hours.
Bound 4: 3 matrices in 3.2s.
Bound 5: 4 matrices in 41s.
Bound 6: 6 matrices in 513s.
Bound 7: ≥ 7 matrices in ≥ 800s.

Bounded block-size for $n = 57$

Using $d_{\min} = 142 \cdot 14^{27}$ (conjectured maxdet):

Bound 2: 2 matrices in 12s (144.50 and 142.02).
Bound 3: 17 matrices in 53 minutes.