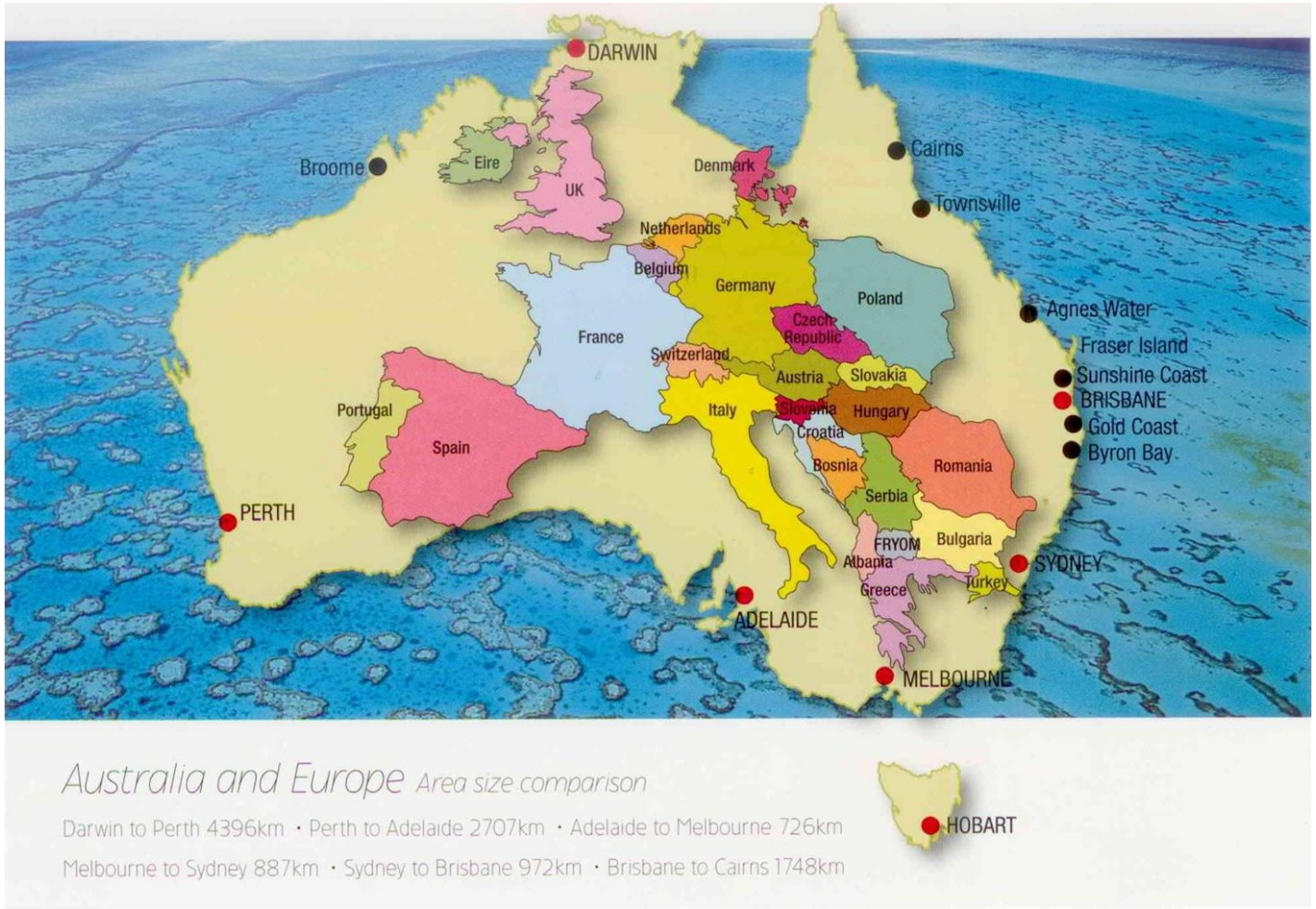
Matroids and MySQL — What to do with big data sets?

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

La Vacquerie-et-Saint-Martin-de-Castries

Australia


DARWIN
Broome
Eire
UK
Denmark
Cairns
Townsville
Netherlands
Belgium
Germany
Poland
Agnes Water
Czech Republic
France
Fraser Island
Switzerland
Sunshine Coast
Austria
Slovakia
BRISBANE
Portugal
Italy
Slovenia
Hungary
Gold Coast
Croatia
Byron Bay
Spain
Bosnia
Romania
Serbia
PERTH
FRYOM
Bulgaria
Albania
SYDNEY
Greece
Turkey
ADELAIDE
MELBOURNE
Australia and Europe Area size comparison
Darwin to Perth 4396km · Perth to Adelaide 2707km · Adelaide to Melbourne 726km
Melbourne to Sydney 887km · Sydney to Brisbane 972km · Brisbane to Cairns 1748km
HOBART

Perth

37TH AUSTRALASIAN CONFERENCE ON COMBINATORIAL MATHEMATICS AND COMBINATORIAL COMPUTING

FEATURED SPEAKERS

Matt DeVos
Primož Potočnik
Graham Farr
Dana Randall
Bill Martin
Tamás Szőnyi
Dillon Mayhew
Nick Wormald

UWA, Perth, December 9th — 14th, 2013

Combinatorial Catalogues

There is a long history of combinatorial mathematicians constructing *lists*, *catalogues* or *census* [sic] of combinatorial objects, such as groups, graphs, designs and geometries.

Some of this work *substantially* predates electronic computers:

- GA Miller's work in the late 1800s on "*substitution groups*"
- The 1930s "Foster Census" of *cubic symmetric graphs*

but it really exploded from the 1960s onwards.

A catalogue of combinatorial geometries

Author(s): John E. Blackburn; Henry H. Crapo; Denis A. Higgs.
Journal: Math. Comp. **27** (1973), 155-166.
MSC: Primary 05B35

Using Catalogues

Although catalogues can only contain information about the *very smallest objects* of any given class, this is often useful.

In addition to the examples and/or counterexamples directly contained in a catalogue, "*small case information*" can be crucial to the development and proof of theoretical results.

- Small cases can identify *interesting cases* for a problem

Using Catalogues

Although catalogues can only contain information about the *very smallest objects* of any given class, this is often useful.

In addition to the examples and/or counterexamples directly contained in a catalogue, "*small case information*" can be crucial to the development and proof of theoretical results.

- Small cases can identify *interesting cases* for a problem
- Small cases can *reveal* the emergence of patterns

Using Catalogues

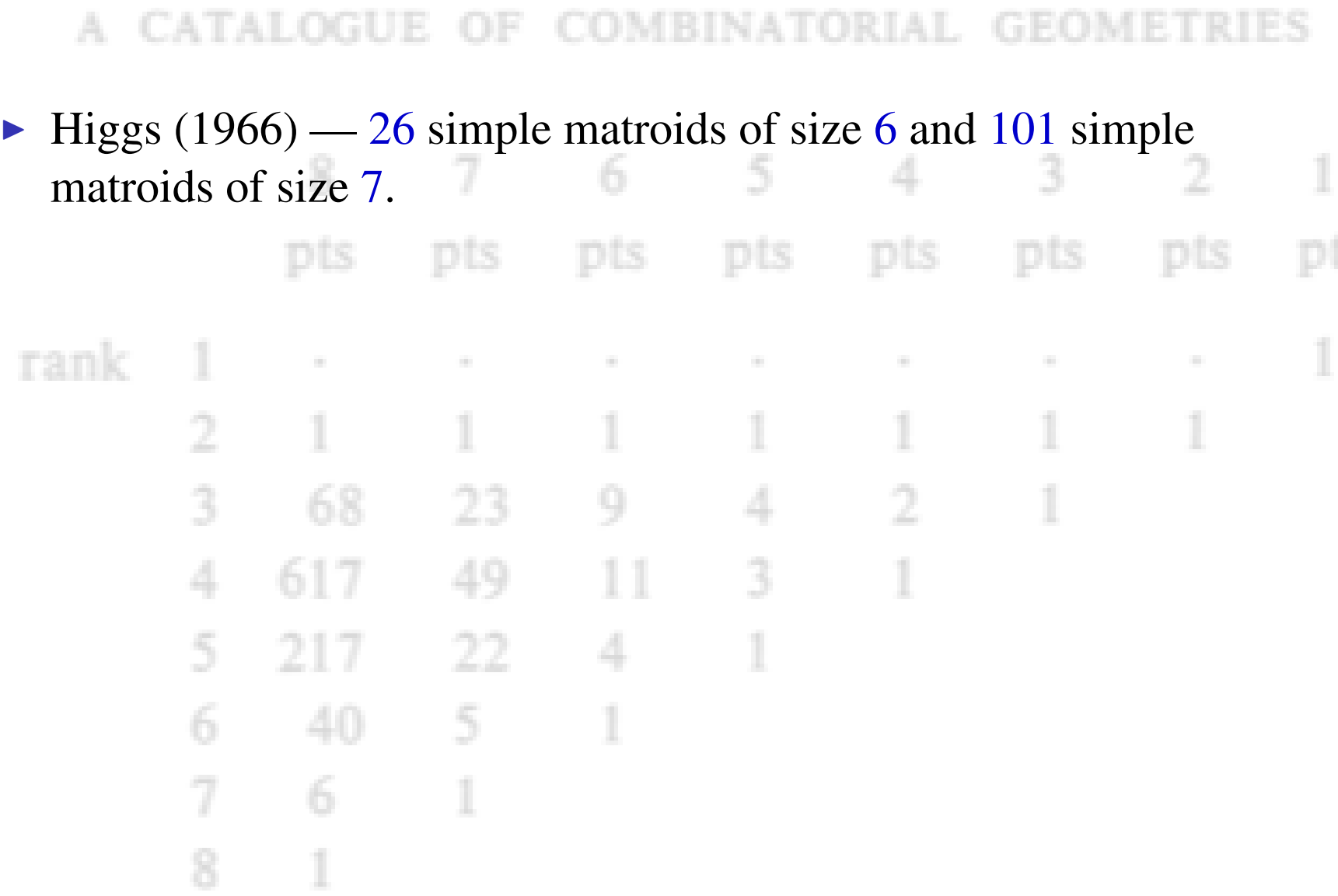
Although catalogues can only contain information about the *very smallest objects* of any given class, this is often useful.

In addition to the examples and/or counterexamples directly contained in a catalogue, "*small case information*" can be crucial to the development and proof of theoretical results.

- Small cases can identify *interesting cases* for a problem
- Small cases can *reveal* the emergence of patterns
- Small cases can *obscure* the emergence of patterns

General Matroids

- Higgs (1966) — 26 simple matroids of size 6 and 101 simple matroids of size 7.



General Matroids

- Higgs (1966) — 26 simple matroids of size 6 and 101 simple matroids of size 7.
- Blackburn, Crapo & Higgs (1973) — 950 simple matroids of size 8.

General Matroids

- Higgs (1966) — 26 simple matroids of size 6 and 101 simple matroids of size 7.
- Blackburn, Crapo & Higgs (1973) — 950 simple matroids of size 8.
- Mayhew & Royle (2009) — 376467 matroids of size 9.

General Matroids

- Higgs (1966) — 26 simple matroids of size 6 and 101 simple matroids of size 7.
- Blackburn, Crapo & Higgs (1973) — 950 simple matroids of size 8.
- Mayhew & Royle (2009) — 376467 matroids of size 9.
- Östergård (2010) — 2630337889305 sparse paving matroids of size 10 and rank 5

At least for the next few years, doing anything other than *counting* the 10-element matroids is well out of reach.

By comparison, *group theorists* have lists of all groups up to about order 2000.

Representable Matroids

The classes of *binary* or *ternary* matroids etc. are much smaller, and so catalogues can pushed further.

- Catalogues of *all* matroids of small enough size, rank
- Catalogues of *minor-closed classes* for slightly larger sizes

Then the question arises:

- What is the best way to *store*, *use* and *provide access* to these catalogues?

Binary and Ternary Matroids

For binary and ternary matroids, additional useful properties hold:

- There is a 1-1 correspondence between *simple matroids* of rank at most d and *subsets of points* of the projective space $PG(d-1,q)$.
- Two matroids are *isomorphic* if and only if the corresponding subsets of points are *equivalent* under the action of the group $PGL(d,q)$ of *linear transformations* of the projective space.

Therefore, if we let Γ be the *point-hyperplane incidence graph* of $PG(d-1,q)$, one technique is to use an *orderly algorithm* to construct subsets of the "*point-vertices*" of Γ that are inequivalent under $\mathrm{Aut}(\Gamma)$.

An orderly algorithm

An *orderly algorithm* is a construction algorithm that generates objects *uniquely up to isomorphism*.

Assume that $\mathcal{L}_k$ contains one representative from each orbit on k-subsets of point-type vertices of Γ.

For each $X \in \mathcal{L}_k$ do the following:

- Compute orbits of $\mathrm{Aut}(\Gamma)_X$
- For each point-type orbit representative x
  - Form $X' = X \cup \{x\}$
  - Compute canonically labelled graph with X' distinguished
  - If x is in the same orbit as the point with lowest canonical label, then accept X' else reject

Then all the accepted $(k+1)$-subsets form $\mathcal{L}_{k+1}$.

Why does this work?

This works because of two key facts:

- The recipe "canonically label and remove lowest numbered element" defines a unique (up to isomorphism) *deconstruction path* for any binary matroid.
  - The *accept/reject* rule for the orderly algorithm ensures that the construction goes — *in reverse* — along this path.
- Adding just one orbit representative guarantees that every matroid is generated only once from its successor on the path.

Numbers of binary matroids

	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$	$r=8$
$e=7$	1	5	8	5	1	
$e=8$		6	15	14	6	1
$e=9$		5	29	38	22	7
$e=10$		4	46	105	80	32
$e=11$		3	64	273	312	151
$e=12$		2	89	700	1285	821
$e=13$		1	112	1794	5632	5098
$e=14$		1	128	4579	26792	37191
$e=15$		1	144	11635	137493	320663
$e=16$			145	29091	745413	3186083
$e=17$			129	70600	4145064	34799393

Binary matroids of rank at most 6

0	1	16	29236	32	46875728	48	11780
1	1	17	70729	33	44065939	49	4708
2	1	18	164818	34	38939167	50	1907
3	2	19	366180	35	32339616	51	791
4	3	20	770299	36	25238329	52	340
5	5	21	1528238	37	18504126	53	155
6	10	22	2852575	38	12742321	54	72
7	19	23	5002828	39	8239590	55	35
8	35	24	8239590	40	5002828	56	19
9	72	25	12742321	41	2852575	57	10
10	155	26	18504126	42	1528238	58	5
11	340	27	25238329	43	770299	59	3
12	791	28	32339616	44	366180	60	2
13	1907	29	38939167	45	164818	61	1
14	4708	30	44065939	46	70729	62	1
15	11780	31	46875728	47	29236	63	1

Counting Binary or Ternary Matroids

We can be confident in the results because the *exact numbers* of simple binary or ternary matroids of any given rank can be determined by *Pólya's Enumeration Theorem*

- Determine the *cycle index* $Z(PGL(d,q))$ of $PGL(d,q)$: The cycle index is the multivariate polynomial in variables $\{X_1, X_2, \ldots\}$ given by

$$Z(G; X_1, X_2, \ldots) := \frac{1}{|G|} \sum_{g \in G} \prod X_k^{n_k(g)}.$$

  where $n_k(g)$ is the number of cycles of length k in g.
- Substitute $1 + x^i$ for each variable X_i.
- The coefficient of x^j is then the number of isomorphism classes of matroids of size j and rank at most d.

Example

In Magma

```
> g := PGammaL(4,2);
> cp := CycleIndexPolynomial(g);
> cp;
1/20160*x[1]^15 + 1/192*x[1]^7*x[2]^4 + 1/96*x[1]^3*x[2]^6 +
    1/16*x[1]^3*x[2]^2*x[4]^2 + 1/18*x[1]^3*x[3]^4 +
    1/6*x[1]*x[2]*x[3]^2*x[6] + 1/8*x[1]*x[2]*x[4]^3 +
    2/7*x[1]*x[7]^2 + 1/180*x[3]^5 + 1/12*x[3]*x[6]^2 +
    1/15*x[5]^3 + 2/15*x[15]
```

Note: For some reason, the `CycleIndexPolynomial` command does not appear in the documentation for Magma. I stumbled on it because it was the obvious name to use.

Rank 4 binary matroids

Substituting $1 + x^i$ for X_i yields the *generating function*

$$x^{15} + x^{14} + x^{13} + 2x^{12} + 3x^{11} + 4x^{10} + 5x^9 + 6x^8 + 6x^7 + 5x^6 + 4x^5 + 3x^4 + 2x^3 + x^2 + x + 1$$

and reading off the coefficients we see that there are *3* binary matroids of size 11 and rank at most 4.

CAN WE PUSH FURTHER?

The exact numbers give us a good idea of where to stop computing!

	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$
$e = 7$	4	19	15	5	1
$e = 8$	3	44	61	26	6
$e = 9$	3	91	277	162	40
$e = 10$	2	199	1439	1381	375
$e = 11$	1	401	8858	17200	5923
$e = 12$	1	806	62311	311580	182059
$e = 13$	1	1504	459828	6876068	9427034
$e = 14$	0	2659	3346151	159373844	608045192
$e = 15$	0	4304	23246482	3609085016	40932394177

Numbers of ternary matroids

SQL

I decided to try storing the data in an actual *relational database* so that it can be queried using *SQL* (Structured Query Language).

Advantages

- Single fixed location for data that is (potentially) accessible from multiple computers
- Relatively easy to integrate web-based access for other users, or even direct programmatic access
- Properties can be added without breaking existing code

Disadvantages

- Rigid format can be limiting
- SQL can be awkward, unnatural, opaque and inefficient

MINIMAL SETUP

At a minimum we want to *store* the matroids and some basic data about them. Thus we create a *database table* called `rank6`.

```
CREATE TABLE binarymatroids (
  id INT PRIMARY KEY,
  size INT,
  rank INT,
  points TEXT);
```

and add the data

```
INSERT INTO binarymatroids VALUES(1,1,1,"1");
INSERT INTO binarymatroids VALUES(2,2,2,"1 2");
INSERT INTO binarymatroids VALUES(3,3,2,"1 2 3");
...
...
```

Minors

To make the database more than mildly useful, it is essential to capture the *minor relationship between* matroids, rather than just properties of single matroids.

It is infeasible to list all the minors for each matroid directly, but the results of *single-element* deletions and contractions can be stored.

```
CREATE TABLE binaryMinors (
  id INT,
  point INT,
  delid INT,
  conid INT);
```

```
CREATE TABLE binaryMinors2 (
  id INT,
  minor INT);
```

To populate this table we need to *find* and *identify* all the single-element deletions and contractions of every matroid.

Minors

To make the database more than mildly useful, it is essential to capture the *minor relationship between* matroids, rather than just properties of single matroids.

It is infeasible to list all the minors for each matroid directly, but the results of *single-element* deletions and contractions can be stored.

```
CREATE TABLE binaryMinors (
  id INT,
  point INT,
  delid INT,
  conid INT);
```

```
CREATE TABLE binaryMinors2 (
  id INT,
  minor INT);
```

To populate this table we need to *find* and *identify* all the single-element deletions and contractions of every matroid.

Identifying a matroid

Each matroid M is assigned a unique *hash code* computed from the graph Γ as follows:

- Use `nauty` to compute the *canonical labelling* of Γ with the points of M distinguished (i.e. "coloured").
- Use a hashing algorithm (in this case SHA) to give each matroid a *fingerprint*.
- Store all fingerprints in lexicographic (alphabetical) order.

```
00000012ac0841740437753a6fa923d41e0e4152 33932540
000000dd31abd8c7f2ddd03d4e8ec7b985d1820b 23143985
000001c4b2413c04851d3fba51d874b04973efe1 05468409
```

An unknown matroid can quickly be identified by computing its fingerprint and then using a *binary search*.

A CANONICAL FORM

As the output of `nauty` can be machine-dependent, it would be better to store a *canonical form* for each matroid.

One candidate would be to take the *lexicographically minimal* matroid equivalent to the given one.

```
gap> LoadPackage("grape");
true
gap> g := PGL(6,2);
<permutation group of size 20158709760 with 2 generators>
gap> m := [1,4,6,9,12,20,22];
[ 1, 4, 6, 9, 12, 20, 22 ]
gap> SmallestImageSet(g,m);
[ 1, 2, 4, 7, 8, 16, 25 ]
```

A CANONICAL FORM

As the output of `nauty` can be machine-dependent, it would be better to store a *canonical form* for each matroid.

One candidate would be to take the *lexicographically minimal* matroid equivalent to the given one.

```
gap> LoadPackage("grape");
true
gap> g := PGL(6,2);
<permutation group of size 20158709760 with 2 generators>
gap> m := [1,4,6,9,12,20,22];
[ 1, 4, 6, 9, 12, 20, 22 ]
gap> SmallestImageSet(g,m);
[ 1, 2, 4, 7, 8, 16, 25 ]
```

But in general, this is *too slow* to compute for large numbers of matroids.

COMPUTING THE MINOR RELATIONSHIP

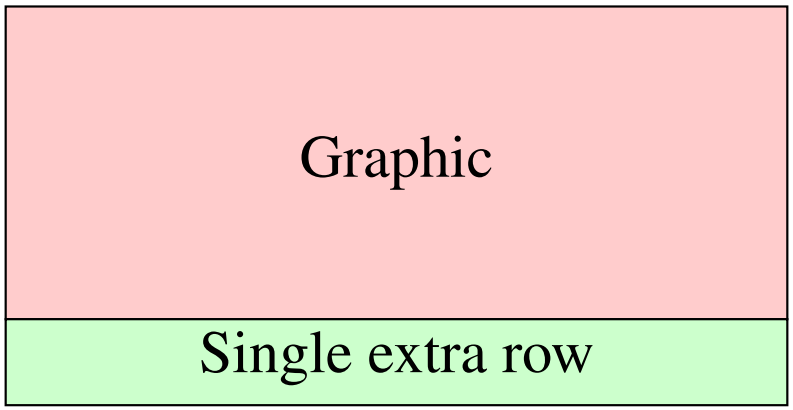
For each matroid/point pair, two minors are *constructed* and *identified* and the information stored in the `binaryMinors` table.

```
mysql> SELECT * FROM binaryMinors WHERE id = 1000;
+------+-------+-------+-------+
| id   | point | delid | conid |
+------+-------+-------+-------+
| 1000 |     1 |   417 |    77 |
| 1000 |     2 |   382 |   125 |
               ..........
               ..........
| 1000 |    26 |   384 |   251 |
| 1000 |    32 |   374 |   309 |
| 1000 |    64 |   374 |   309 |
| 1000 |   125 |   374 |   309 |
+------+-------+-------+-------+
```

This is a *huge computation* due to the sheer number of matroids but is only done once.

SAMPLE USE

An *even cycle matroid* is a binary matroid M that with a representing matrix of the form



Equivalently M is even cycle if there is binary element e such that $(M + e)/e$ is graphic.

This property is minor-closed — what are the *excluded minors* for even cycle matroids?

SETTING UP THE DATABASE

We'll add the property `isEvenCycle` to the database

```
ALTER TABLE binarymatroids
   ADD COLUMN (isEvenCycle tinyint(1));
```

Then we actually have to compute which matroids *are* even cycle.

```
def is_evencycle(m):
    return True in [n.contract('X').is_graphic()
                    for n in m.linear_extensions('X')]
```

and enter this data into the database:

```
....
UPDATE binarymatroids SET isEvenCycle = 1 WHERE id = 102323;
UPDATE binarymatroids SET isEvenCycle = 1 WHERE id = 102324;
UPDATE binarymatroids SET isEvenCycle = 0 WHERE id = 102325;
.....
```

THE FINAL COMMAND

```
SELECT B.id, B.points FROM binarymatroids B
  WHERE B.size < 17 AND B.isEvenCycle = 0
  AND NOT EXISTS
     (SELECT * FROM binarymatroids B2, binaryminors2 BM
        WHERE B.id = BM.id AND
            BM.minor = B2.id AND B2.isEvenCycle = 0);
```

In pictures

id	isEC
⋮	⋮
233	0
234	0
235	0
236	0
⋮	⋮

B

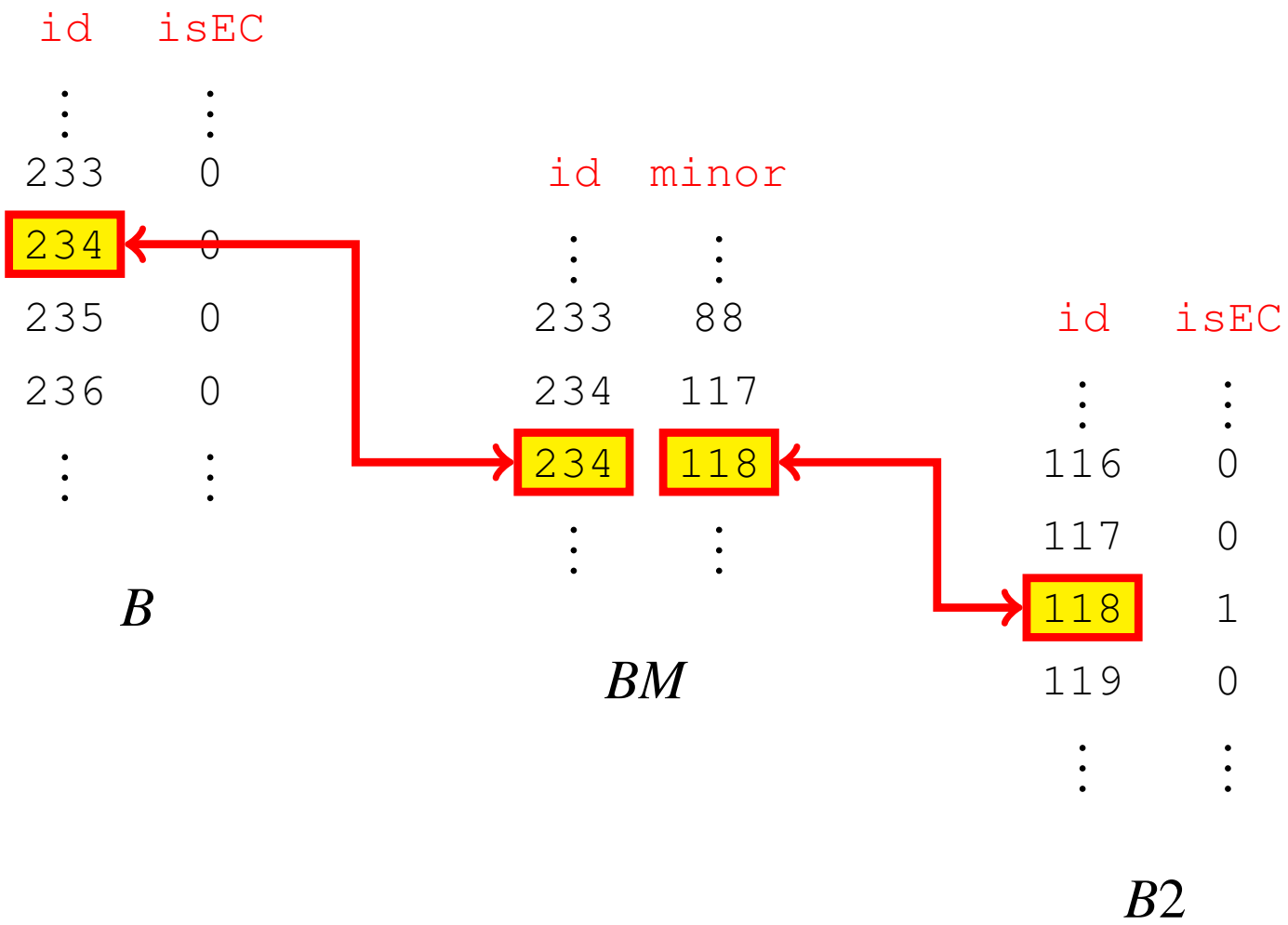
id	minor
⋮	⋮
233	88
234	117
234	118
⋮	⋮

BM

id	isEC
⋮	⋮
116	0
117	0
118	1
119	0
⋮	⋮

$B2$

In pictures


id isEC
233 0
234 0
235 0
236 0
B
id minor
233 88
234 117
234 118
BM
id isEC
116 0
117 0
118 1
119 0
B2

And the final

```
+---------+-----------------------------------------------------+
| id      | points                                              |
+---------+-----------------------------------------------------+
|     748 | 1 2 3 4 8 16 29 32 46 52 59                         |
|     774 | 1 2 4 8 15 16 21 32 41 51 63                        |
|    1343 | 1 2 3 4 5 8 9 14 16 22 26 29                        |
|    1346 | 1 2 4 7 8 11 13 14 16 21 25 31                      |...
...
...
| 4397271 | 1 2 4 8 16 18 28 32 64 66 127 128 133 143 179 208   |
| 4397878 | 1 2 4 8 16 28 32 50 64 80 127 128 133 143 179 208   |
| 4397915 | 1 2 4 8 10 16 28 32 51 64 127 128 133 143 179 208   |
| 4398085 | 1 2 4 5 8 16 28 32 64 67 127 128 133 143 179 208    |
| 4401402 | 1 2 4 8 9 16 32 64 127 128 130 133 139 143 179 208  |
| 4401801 | 1 2 4 8 9 16 32 54 64 127 128 131 133 143 179 208   |
| 4401823 | 1 2 4 8 9 16 32 64 127 128 131 133 143 178 179 208  |
| 4434811 | 1 2 4 7 8 16 32 39 64 65 127 128 143 148 179 205    |
| 4435762 | 1 2 4 7 8 16 32 64 125 127 128 143 148 156 179 205  |
| 4438645 | 1 2 4 8 12 16 24 32 64 125 127 128 143 148 179 205  |
| 4446827 | 1 2 4 8 16 32 44 64 65 127 128 143 148 179 180 205  |
| 4506185 | 1 2 4 5 8 16 32 64 127 128 131 136 141 143 179 213  |
| 4507486 | 1 2 4 8 16 32 64 66 127 128 131 143 160 179 197 213 |
+---------+-----------------------------------------------------+
717 rows in set (2 min 36.41 sec)
```

How can this be provided in Sage?

Demo

Problems

Many problems:

- Technical problems
- Non-technical problems
  Too much data, but not enough proofs!