

Fokko du Cloux' `atlas` software and the recent E_8 computation

Marc van Leeuwen

Laboratoire de Mathématiques et Applications
Université de Poitiers

Friday 30 November 2007
DIAMANT/EIDMA symposium Soesterberg

or

How the work of Fokko du Cloux
made it possible to compute
Kazhdan-Lusztig-Vogan polynomials,
even those
for the big block
of the split real form
of the complex reductive Lie group
of type E_8

- 1 Computational Lie theory and the Atlas project
- 2 The `atlas` program
- 3 The case $E_8(\mathbf{R})$
- 4 Modifying the `atlas` program for E_8
- 5 Actually running the E_8 computations

- 1 Computational Lie theory and the Atlas project
- 2 The `atlas` program
- 3 The case $E_8(\mathbf{R})$
- 4 Modifying the `atlas` program for E_8
- 5 Actually running the E_8 computations

Before the Atlas project

- 1981: first Kazhdan-Lusztig computations (Mark Goresky)
- 1986: Dean Alvis computes KL polynomials up to H_4
- about 1987–1995: development of the program LiE (by a team in Amsterdam headed by Arjeh Cohen)
 - Character computations
 - Finite dimensional representation theory for complex groups
- 1991–Feb 2004: development of the program Coxeter (by Fokko du Cloux)
 - Coxeter group computations
 - Bruhat order
 - Kazhdan-Lusztig polynomials
- June 1992: Conference at Lyon co-organised by Fokko du Cloux “Coxeter groups and computational representation theory”
- May-June 2002: workshop at Montreal (organised by Bill Casselman and Friedrich Knop) “Workshop on Computational Lie Theory”

Before the Atlas project

The next important program in the field is the one written by Dean Alvis, then at MIT, around 1986. Neither the code nor the results of this program have ever been made publicly available, but its results have been used by several authors to settle computationally some questions on exceptional Lie groups. Also, several results proved for crystallographic groups have been checked for the non-crystallographic cases (the hard one being type H_4) by this program: for instance, Alvis has checked the positivity of the coefficients of the $P_{x,y}$, and determined the left cells in type H_4 in [1]. In my opinion, considering the hardware available at the time, this is a very remarkable achievement that has not been sufficiently appreciated; as far as I know, the only other program that has ever computed all the Kazhdan-Lusztig polynomials in type H_4 is my own program [7], with the advantage of much better machines – and it's still one of the really hard cases.

Fokko du Cloux, in “The State of the Art in the Computation of Kazhdan-Lusztig polynomials”, 1996

Before the Atlas project

- 1981: first Kazhdan-Lusztig computations (Mark Goresky)
- 1986: Dean Alvis computes KL polynomials up to H_4
- about 1987–1995: development of the program LiE (by a team in Amsterdam headed by Arjeh Cohen)
 - Character computations
 - Finite dimensional representation theory for complex groups
- 1991–Feb 2004: development of the program Coxeter (by Fokko du Cloux)
 - Coxeter group computations
 - Bruhat order
 - Kazhdan-Lusztig polynomials
- June 1992: Conference at Lyon co-organised by Fokko du Cloux “Coxeter groups and computational representation theory”
- May-June 2002: workshop at Montreal (organised by Bill Casselman and Friedrich Knop) “Workshop on Computational Lie Theory”

Before the Atlas project

- 1981: first Kazhdan-Lusztig computations (Mark Goresky)
- 1986: Dean Alvis computes KL polynomials up to H_4
- about 1987–1995: development of the program LiE (by a team in Amsterdam headed by Arjeh Cohen)
 - Character computations
 - Finite dimensional representation theory for complex groups
- 1991–Feb 2004: development of the program Coxeter (by Fokko du Cloux)
 - Coxeter group computations
 - Bruhat order
 - Kazhdan-Lusztig polynomials
- June 1992: Conference at Lyon co-organised by Fokko du Cloux “Coxeter groups and computational representation theory”
- May-June 2002: workshop at Montreal (organised by Bill Casselman and Friedrich Knop) “Workshop on Computational Lie Theory”

4.3. Let us now turn to the present. As far as I know, [7] is the best current program in the subject, both in range and in speed. It is made available with some reluctance, for its performance is good, but the actual code is nowhere near any reasonable standard of “readability” that would make it easy, say, for an interested user to understand the structure of the program and introduce his or her own improvements or modifications;

5.4. Finally, another direction for the future would be the computation of the “Kazhdan-Lusztig polynomials” associated to the theory of Harish-Chandra modules for a real reductive Lie group G . Here the combinatorial difficulties increase by several orders of magnitude, and it is not even clear (at least to this author) how to formulate the algorithm in a way that a computer will understand. Although there is some activity going on in this direction, I am not aware of any running program at the time of writing.

Fokko du Cloux, in “The State of the Art in the Computation of Kazhdan-Lusztig polynomials”, 1996

The Atlas project

At the 2002 Computational Lie Theory workshop, the first plans for the Atlas project were made.

- Goal: making explicitly available the state of the art in the computational theory of Real Lie groups and their infinite dimensional representations, in the spirit of the “Atlas of Finite Groups” (Conway, Parker, Norton, 1985).
- Essential part: implementing “known” algorithmic descriptions.
- Ultimate goal: determine unitary dual of any real reductive group.

Recruiting a programmer

Dear Marc,

I was in Palo Alto end of July, to participate in a workshop that had as goal setting up a project to do the (infinite dimensional) theory of real and p -adic Lie groups by computer. One of the principal goals would be determining the unitary irreducible representations, in particular for exceptional groups.

The main participants were Vogan, Barbasch, Adams, Stembridge, and Trapa, that I believe were all also present in Montreal, when we both were there last year. For the theory of the project you should not come to me [...]

Don't be scared by the fact that you probably don't know very much about this subject; at this point I don't either, even though in the course of time I have heard many talks concerning it, so that I am acquainted with most of the words. [...]

(message from Fokko, 25 August 2003)

Members of the Atlas project

Jeffrey Adams
Dan Barbasch
Birne Binengar
Bill Casselman
Dan Ciubotaru
Fokko du Cloux
Scott Crofts
Tatiana Howard
Alfred Noel
Marc van Leeuwen

Alessandra Pantano
Annegret Paul
Patrick Polo
Siddhartha Sahi
Susana Salamanca
John Stembridge
Peter Trapa
David Vogan
Wai-Ling Yee
Jiu-Kang Yu

Members of the Atlas project

Jeffrey Adams

Dan Barbasch

Birne Binengar

Bill Casselman

Dan Ciubotaru

Fokko du Cloux

Scott Crofts

Tatiana Howard

Alfred Noel

Marc van Leeuwen

Alessandra Pantano

Annegret Paul

Patrick Polo

Siddhartha Sahi

Susana Salamanca

John Stembridge

Peter Trapa

David Vogan

Wai-Ling Yee

Jiu-Kang Yu

Events pertinent to `atlas` development

Some dates (personal selection):

- 2002. 'Atlas of Lie groups and Representations' project conceived.
- 2003. First of the annual Atlas workshops at Palo Alto.
- Sept 2003. Meeting at Berder. Fokko is preparing `atlas`.
- February 2004. Fokko finishes `Coxeter 3.0` and starts programming of the `atlas` software.
- September 2004. Fokko works on structure theory.
- September 2005. Fokko visits Poitiers; is working on $K \backslash G/B$.
- November 2005. `atlas` can compute KLV polynomials, W -cells.
- November 2005. Fokko is diagnosed with ALS.
- early 2006. Fokko cannot use a keyboard, but continues to work.
- 10 November 2006. Death of Fokko du Cloux.
- 8 January 2007. `atlas` computes KLV polynomials for $E_8(\mathbf{R})$.
- 19 March 2007. Public announcement of the above result.
- 20?? Algorithm for branching to K implemented.

- 1 Computational Lie theory and the Atlas project
- 2 The atlas program**
- 3 The case $E_8(\mathbf{R})$
- 4 Modifying the atlas program for E_8
- 5 Actually running the E_8 computations

The atlas software

It is (currently) a stand-alone command-line program written in C++.

It contains:

- A library of mathematical data types, with associated functions
- Output functions for sending results to screen/file
- An interactive system for specifying a computation
- A help system describing commands and value encoding
- Infrastructure necessary for implementing the above

The infrastructure and library can be used in other contexts:

- A user programming environment
- Interfacing to existing Computer Algebra systems

The atlas software

It is (currently) a stand-alone command-line program written in C++.

It contains:

- A library of mathematical data types, with associated functions
- Output functions for sending results to screen/file
- An interactive system for specifying a computation
- A help system describing commands and value encoding
- Infrastructure necessary for implementing the above

The infrastructure and library can be used in other contexts:

- A user programming environment
- Interfacing to existing Computer Algebra systems

Components of the `atlas` mathematical library

Many layers of software implementation (everything is *discrete*)

- Root systems
- Root data
- Weyl group and Tits group
- Inner classes of real forms (Dynkin diagram involutions)
- Real Cartan subgroups (root datum involutions) and Weyl groups
- Real forms (weak and strong)
- One sided parameter sets: $K \backslash G/B$
- Two sided parameter sets: blocks
- Kazhdan-Lusztig-Vogan polynomials
- W -cells

Components of the `atlas` mathematical library

Many layers of software implementation (everything is *discrete*)

- Root systems
- Root data
- Weyl group and Tits group
- Inner classes of real forms (Dynkin diagram involutions)
- Real Cartan subgroups (root datum involutions) and Weyl groups
- Real forms (weak and strong)
- One sided parameter sets: $K \backslash G/B$
- Two sided parameter sets: blocks
- Kazhdan-Lusztig-Vogan polynomials
- W -cells

Many layers of software implementation (everything is *discrete*)

- Root systems
- Root data
- Weyl group and Tits group
- Inner classes of real forms (Dynkin diagram involutions)
- Real Cartan subgroups (root datum involutions) and Weyl groups
- Real forms (weak and strong)
- One sided parameter sets: $K \backslash G/B$
- Two sided parameter sets: blocks
- Kazhdan-Lusztig-Vogan polynomials
- W -cells

Computation of KLV polynomials

- Point of departure: the block
 - A finite combinatorially constructed set,
 - carrying a grading ℓ (length)
 - and transition relations between elements (cross and Cayley).
- Must compute: a triangular matrix with entries in $\mathbf{Z}[q]$, its rows and columns are labelled by block elements.
- The entries $P_{x,y} \in \mathbf{Z}[q]$ can be computed recursively.
- For $\ell(x) < \ell(y)$ one has $\deg P_{x,y} \leq \frac{\ell(y) - \ell(x) - 1}{2}$.
- The recursion relations are additive, with in addition the occasional use of a scalar $\mu(x', y')$ defined as $\text{coef}(q^{\frac{\ell(y') - \ell(x') - 1}{2}}, P_{x', y'})$.
- The nature of the recursion forces computing everything at once, and keeping all polynomials in memory

Computation of KLV polynomials

- Point of departure: the block
 - A finite combinatorially constructed set,
 - carrying a grading ℓ (length)
 - and transition relations between elements (cross and Cayley).
- Must compute: a triangular matrix with entries in $\mathbf{Z}[q]$, its rows and columns are labelled by block elements.
- The entries $P_{x,y} \in \mathbf{Z}[q]$ can be computed recursively.
- For $\ell(x) < \ell(y)$ one has $\deg P_{x,y} \leq \frac{\ell(y) - \ell(x) - 1}{2}$.
- The recursion relations are additive, with in addition the occasional use of a scalar $\mu(x', y')$ defined as $\text{coef}(q^{\frac{\ell(y') - \ell(x') - 1}{2}}, P_{x', y'})$.
- The nature of the recursion forces computing everything at once, and keeping all polynomials in memory

Computation of KLV polynomials

- Point of departure: the block
 - A finite combinatorially constructed set,
 - carrying a grading ℓ (length)
 - and transition relations between elements (cross and Cayley).
- Must compute: a triangular matrix with entries in $\mathbf{Z}[q]$, its rows and columns are labelled by block elements.
- The entries $P_{x,y} \in \mathbf{Z}[q]$ can be computed recursively.
- For $\ell(x) < \ell(y)$ one has $\deg P_{x,y} \leq \frac{\ell(y) - \ell(x) - 1}{2}$.
- The recursion relations are additive, with in addition the occasional use of a scalar $\mu(x', y')$ defined as $\text{coef}(q^{\frac{\ell(y') - \ell(x') - 1}{2}}, P_{x', y'})$.
- The nature of the recursion forces computing everything at once, and keeping all polynomials in memory

Computation of KLV polynomials

- Point of departure: the block
 - A finite combinatorially constructed set,
 - carrying a grading ℓ (length)
 - and transition relations between elements (cross and Cayley).
- Must compute: a triangular matrix with entries in $\mathbf{Z}[q]$, its rows and columns are labelled by block elements.
- The entries $P_{x,y} \in \mathbf{Z}[q]$ can be computed recursively.
- For $\ell(x) < \ell(y)$ one has $\deg P_{x,y} \leq \frac{\ell(y) - \ell(x) - 1}{2}$.
- The recursion relations are additive, with in addition the occasional use of a scalar $\mu(x', y')$ defined as $\text{coef}(q^{\frac{\ell(y') - \ell(x') - 1}{2}}, P_{x', y'})$.
- The nature of the recursion forces computing everything at once, and keeping all polynomials in memory

- 1 Computational Lie theory and the Atlas project
- 2 The `atlas` program
- 3 The case $E_8(\mathbf{R})$**
- 4 Modifying the `atlas` program for E_8
- 5 Actually running the E_8 computations

Why E_8 ?

What is so special about E_8 ?

- It is (by far) the largest exceptional type.
- It's root system and associated data are exceptionally dense.
- The real form $E_8(\mathbf{R})$ has blocks of sizes 1, 71654, and 453060.
- The big block of $E_8(\mathbf{R})$ is *much* more complicated than blocks of comparable size in classical types.
- It remained the only important case **atlas** could not complete.

Comparison: for $SL(11, \mathbf{R})$ (A_{10}) large block of size 173712 has

- 41.395.956 nonzero entries (8%) to compute, containing
- 60.228 distinct polynomials, with
- 394.415 coefficients, all ≤ 150 ,
- 0,5 GB of memory required.

The big block of the *middle* real form for E_8 has size 71654, with

- 63.343.253 nonzero entries (49%) to compute, containing
- 10.147.581 distinct polynomials, with
- 101.229.186 coefficients, all ≤ 2545 ,
- 1,5 GB of memory required.

Why E_8 ?

What is so special about E_8 ?

- It is (by far) the largest exceptional type.
- It's root system and associated data are exceptionally dense.
- The real form $E_8(\mathbf{R})$ has blocks of sizes 1, 71654, and 453060.
- The big block of $E_8(\mathbf{R})$ is *much* more complicated than blocks of comparable size in classical types.
- It remained the only important case **atlas** could not complete.

Comparison: for $SL(11, \mathbf{R})$ (A_{10}) large block of size 173712 has

- 41.395.956 nonzero entries (8%) to compute, containing
- 60.228 distinct polynomials, with
- 394.415 coefficients, all ≤ 150 ,
- 0,5 GB of memory required.

The big block of the *middle* real form for E_8 has size 71654, with

- 63.343.253 nonzero entries (49%) to compute, containing
- 10.147.581 distinct polynomials, with
- 101.229.186 coefficients, all ≤ 2545 ,
- 1,5 GB of memory required.

Why E_8 ?

What is so special about E_8 ?

- It is (by far) the largest exceptional type.
- It's root system and associated data are exceptionally dense.
- The real form $E_8(\mathbf{R})$ has blocks of sizes 1, 71654, and 453060.
- The big block of $E_8(\mathbf{R})$ is *much* more complicated than blocks of comparable size in classical types.
- It remained the only important case **atlas** could not complete.

Comparison: for $SL(11, \mathbf{R})$ (A_{10}) large block of size 173712 has

- 41.395.956 nonzero entries (8%) to compute, containing
- 60.228 distinct polynomials, with
- 394.415 coefficients, all ≤ 150 ,
- 0,5 GB of memory required.

The big block of the *middle* real form for E_8 has size 71654, with

- 63.343.253 nonzero entries (49%) to compute, containing
- 10.147.581 distinct polynomials, with
- 101.229.186 coefficients, all ≤ 2545 ,
- 1,5 GB of memory required.

Why E_8 ?

What is so special about E_8 ?

- It is (by far) the largest exceptional type.
- It's root system and associated data are exceptionally dense.
- The real form $E_8(\mathbf{R})$ has blocks of sizes 1, 71654, and 453060.
- The big block of $E_8(\mathbf{R})$ is *much* more complicated than blocks of comparable size in classical types.
- It remained the only important case **atlas** could not complete.

Comparison: for $SL(11, \mathbf{R})$ (A_{10}) large block of size 173712 has

- 41.395.956 nonzero entries (8%) to compute, containing
- 60.228 distinct polynomials, with
- 394.415 coefficients, all ≤ 150 ,
- 0,5 GB of memory required.

The big block of the *middle* real form for E_8 has size 71654, with

- 63.343.253 nonzero entries (49%) to compute, containing
- 10.147.581 distinct polynomials, with
- 101.229.186 coefficients, all ≤ 2545 ,
- 1,5 GB of memory required.

Why E_8 ?

What is so special about E_8 ?

- It is (by far) the largest exceptional type.
- It's root system and associated data are exceptionally dense.
- The real form $E_8(\mathbf{R})$ has blocks of sizes 1, 71654, and 453060.
- The big block of $E_8(\mathbf{R})$ is *much* more complicated than blocks of comparable size in classical types.
- It remained the only important case **atlas** could not complete.

Comparison: for $SL(11, \mathbf{R})$ (A_{10}) large block of size 173712 has

- 41.395.956 nonzero entries (8%) to compute, containing
- 60.228 distinct polynomials, with
- 394.415 coefficients, all ≤ 150 ,
- 0,5 GB of memory required.

The big block of the *middle* real form for E_8 has size 71654, with

- 63.343.253 nonzero entries (49%) to compute, containing
- 10.147.581 distinct polynomials, with
- 101.229.186 coefficients, all ≤ 2545 ,
- 1,5 GB of memory required.

How hard is doing the big block of $E_8(\mathbf{R})$

Before we could actually do the calculation, we knew or estimated:

- There are **6.083.625.251** entries to compute.
- If about **50%** are nonzero, that is about **3 billion** polynomials.
- Each matrix entry needs **12 bytes**:
in all **36 GB** just to identify matrix entries.
- Each polynomial has at most **32** coefficients (often much less).
- The largest coefficient *exceeds* $2^{16} = 65536$; so computation without overflow requires **4 bytes** per coefficient.
- Depending on number of *distinct polynomials*, estimated coefficient storage needed is **5 GB – 100 GB**.
- There is **important memory overhead** in storing polynomials.

Moreover the textual output format is not at all suited to such enormous amounts of data.

How hard is doing the big block of $E_8(\mathbf{R})$

Before we could actually do the calculation, we knew or estimated:

- There are **6.083.625.251** entries to compute.
- If about **50%** are nonzero, that is about **3 billion** polynomials.
- Each matrix entry needs **12 bytes**:
in all **36 GB** just to identify matrix entries.
- Each polynomial has at most **32** coefficients (often much less).
- The largest coefficient *exceeds* $2^{16} = 65536$; so
computation without overflow requires **4 bytes** per coefficient.
- Depending on number of *distinct polynomials*, estimated
coefficient storage needed is **5 GB – 100 GB**.
- There is **important memory overhead** in storing polynomials.

Moreover the textual output format is not at all suited to such enormous amounts of data.

How hard is doing the big block of $E_8(\mathbf{R})$

Before we could actually do the calculation, we knew or estimated:

- There are **6.083.625.251** entries to compute.
- If about **50%** are nonzero, that is about **3 billion** polynomials.
- Each matrix entry needs **12 bytes**:
in all **36 GB** just to identify matrix entries.
- Each polynomial has at most **32** coefficients (often much less).
- The largest coefficient *exceeds* $2^{16} = 65536$; so
computation without overflow requires **4 bytes** per coefficient.
- Depending on number of *distinct polynomials*, estimated
coefficient storage needed is **5 GB – 100 GB**.
- There is **important memory overhead** in storing polynomials.

Moreover the textual output format is not at all suited to such enormous amounts of data.

How hard is doing the big block of $E_8(\mathbf{R})$

Before we could actually do the calculation, we knew or estimated:

- There are **6.083.625.251** entries to compute.
- If about **50%** are nonzero, that is about **3 billion** polynomials.
- Each matrix entry needs **12 bytes**:
in all **36 GB** just to identify matrix entries.
- Each polynomial has at most **32** coefficients (often much less).
- The largest coefficient *exceeds* $2^{16} = 65536$; so
computation without overflow requires **4 bytes** per coefficient.
- Depending on number of *distinct polynomials*, estimated
coefficient storage needed is **5 GB – 100 GB**.
- There is **important memory overhead** in storing polynomials.

Moreover the textual output format is not at all suited to such enormous amounts of data.

Back to reality

The computer with largest central memory to which we had access was **sage**, at the University of Washington (Seattle):

64 GB of RAM Thank you William Stein!

But the numbers that emerged for E_8 do not justify optimism:

- 3.393.819.896 nonzero entries (56%) to compute
- 1.181.642.979 distinct polynomials
- 13.721.641.221 coefficients
- maximal coefficient 11.808.808

To do the computation with the existing software would require about **160 GB** of RAM:

- 40 GB for the matrix
- 55 GB for brute storage of the polynomial coefficients
- et 65 GB for various internal structures (mostly to locate and represent polynomials).

Back to reality

The computer with largest central memory to which we had access was **sage**, at the University of Washington (Seattle):

64 GB of RAM **Thank you William Stein!**

But the numbers that emerged for E_8 do not justify optimism:

- 3.393.819.896 nonzero entries (56%) to compute
- 1.181.642.979 distinct polynomials
- 13.721.641.221 coefficients
- maximal coefficient 11.808.808

To do the computation with the existing software would require about **160 GB** of RAM:

- 40 GB for the matrix
- 55 GB for brute storage of the polynomial coefficients
- et 65 GB for various internal structures (mostly to locate and represent polynomials).

Back to reality

The computer with largest central memory to which we had access was **sage**, at the University of Washington (Seattle):

64 GB of RAM Thank you William Stein!

But the numbers that emerged for E_8 do not justify optimism:

- **3.393.819.896** nonzero entries (**56%**) to compute
- **1.181.642.979** distinct polynomials
- **13.721.641.221** coefficients
- maximal coefficient **11.808.808**

To do the computation with the existing software would require about **160 GB** of RAM:

- **40** GB for the matrix
- **55** GB for brute storage of the polynomial coefficients
- et **65** GB for various internal structures (mostly to locate and represent polynomials).

Back to reality

The computer with largest central memory to which we had access was **sage**, at the University of Washington (Seattle):

64 GB of RAM Thank you William Stein!

But the numbers that emerged for E_8 do not justify optimism:

- **3.393.819.896** nonzero entries (**56%**) to compute
- **1.181.642.979** distinct polynomials
- **13.721.641.221** coefficients
- maximal coefficient **11.808.808**

To do the computation with the existing software would require about **160 GB** of RAM:

- **40 GB** for the matrix
- **55 GB** for brute storage of the polynomial coefficients
- et **65 GB** for various internal structures (mostly to locate and represent polynomials).

Overview

- 1 Computational Lie theory and the Atlas project
- 2 The `atlas` program
- 3 The case $E_8(\mathbf{R})$
- 4 Modifying the `atlas` program for E_8**
- 5 Actually running the E_8 computations

Reduce modulo, and conquer

Computing in $(\mathbf{Z}/n)[q]$ instead of $\mathbf{Z}[q]$ will give the *correct* polynomials, but with coefficients reduced modulo n .

Because: although $\mu(x, y)$ is the leading coefficient of $P_{x,y}$ if nonzero, it is always its coefficient of the *fixed* degree $\frac{\ell(y) - \ell(x) - 1}{2}$.

Therefore one can perform independent modular computations, and combine the results for coefficients modulo the least common multiple of the moduli (Chinese Remainder Theorem).

If moduli ≤ 256 are used, each modular coefficient needs only 1 byte.

Reduce modulo, and conquer

Computing in $(\mathbf{Z}/n)[q]$ instead of $\mathbf{Z}[q]$ will give the *correct* polynomials, but with coefficients reduced modulo n .

Because: although $\mu(x, y)$ is the leading coefficient of $P_{x,y}$ if nonzero, it is always its coefficient of the *fixed* degree $\frac{\ell(y) - \ell(x) - 1}{2}$.

Therefore one can perform independent modular computations, and combine the results for coefficients modulo the least common multiple of the moduli (Chinese Remainder Theorem).

If moduli ≤ 256 are used, each modular coefficient needs only 1 byte.

Reduce modulo, and conquer

Computing in $(\mathbf{Z}/n)[q]$ instead of $\mathbf{Z}[q]$ will give the *correct* polynomials, but with coefficients reduced modulo n .

Because: although $\mu(x, y)$ is the leading coefficient of $P_{x,y}$ if nonzero, it is always its coefficient of the *fixed* degree $\frac{\ell(y) - \ell(x) - 1}{2}$.

Therefore one can perform independent modular computations, and combine the results for coefficients modulo the least common multiple of the moduli (Chinese Remainder Theorem).

If moduli ≤ 256 are used, each modular coefficient needs only 1 byte.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of sage;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbf{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of sage;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbf{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of sage;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbf{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of **sage**;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbf{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of **sage**;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbb{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Other modifications of the software

Modular reduction saves memory, but not enough (about 40 of 160 GB). Other changes were made:

- a search tree for identifying polynomials is replaced by a hash table;
- in the matrix 8-byte pointers can be replaced by a 4-byte identification number of the polynomial;
- all polynomial coefficients can be stored in one coefficient pool;
- computation can be split among threads, to better employ the 16 processors of `sage`;
- binary output routines replace textual ones.

With all changes, the computation for the big block of $E_8(\mathbf{R})$ needs about 55 GB of RAM.

Small programs `matrix-merge` and `coef-merge` were written, to read back and combine the binary files produced in the modular runs.

Overview

- 1 Computational Lie theory and the Atlas project
- 2 The `atlas` program
- 3 The case $E_8(\mathbf{R})$
- 4 Modifying the `atlas` program for E_8
- 5 Actually running the E_8 computations

An obstacle course

- Tuesday 19 Dec 2006. First **complete run** mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds. Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds. Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run **terminates successfully!**
Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds.
Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs.
Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully!
Computation mod 255 started; **crash**.
- Wednesday 27. New try mod 255 succeeds.
Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs.
Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 **succeeds**.
Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds. Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds. Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 **succeeds**. `matrix-merge` started, it **succeeds** in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. System crash. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

An obstacle course

- Tuesday 19 Dec 2006. First complete run mod 251 (in 17 hours).
- Thursday 21. David Vogan finds and repairs bugs in the binary output routines.
- Friday 22. Computation mod 256 started. Crash at $y = 452274$. Recombination programs ready. Computation mod 256 restarted.
- Saturday 23. Mod 256 run terminates successfully! Computation mod 255 started; crash.
- Wednesday 27. New try mod 255 succeeds. Computation mod 253 crashes at $y = 398305$.
- Wednesday 3 Jan 2007. Computation mod 253 restarted.
- Thursday 4. Mod 253 succeeds. `matrix-merge` started, it succeeds in 9 hours. `coef-merge` and a mod 251 run started.
- Friday 5, evening. **System crash**. William Stein replaces hard disk with a backup with data of mod 256, 255, and 253 runs. Restart of `coef-merge` for these moduli.

Conclusion

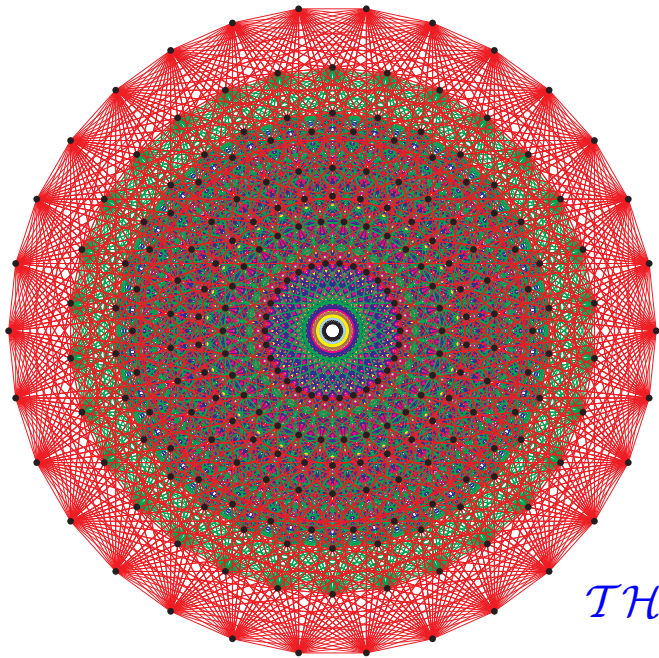
- Saturday 6, morning. Minor update of `coef-merge`.
- afternoon. David Vogan signals `coef-merge` output is **28GB too large**
Meanwhile he recovers mod **251** data from original disk, and reruns `matrix-merge`.
- evening. I can locate point where `coef-merge` output goes wrong, but cannot find bug.
Start test run to diagnose point of error.
- Sunday 7, early morning. Test run did *not* go wrong.
Bug had inadvertently been repaired already.
- noon. David restarts `coef-merge` for **4** moduli.
- Monday 8 Jan 2007, 14:57 CET. `coef-merge` terminates after **27** hours of computation.

Conclusion

- Saturday 6, morning. Minor update of `coef-merge`.
- afternoon. David Vogan signals `coef-merge` output is 28GB too large
Meanwhile he recovers mod 251 data from original disk, and reruns `matrix-merge`.
- evening. I can locate point where `coef-merge` output goes wrong, but cannot find bug.
Start test run to diagnose point of error.
- Sunday 7, early morning. Test run did *not* go wrong.
Bug had inadvertently been repaired already.
- noon. David restarts `coef-merge` for 4 moduli.
- Monday 8 Jan 2007, 14:57 CET. `coef-merge` terminates after 27 hours of computation.

Conclusion

- Saturday 6, morning. Minor update of `coef-merge`.
- afternoon. David Vogan signals `coef-merge` output is 28GB too large
Meanwhile he recovers mod 251 data from original disk, and reruns `matrix-merge`.
- evening. I can locate point where `coef-merge` output goes wrong, but cannot find bug.
Start test run to diagnose point of error.
- Sunday 7, early morning. Test run did *not* go wrong.
Bug had inadvertently been repaired already.
- noon. David restarts `coef-merge` for 4 moduli.
- Monday 8 Jan 2007, 14:57 CET. `coef-merge` terminates after 27 hours of computation.



THE END