Marc van Leeuwen
Université de Poitiers, Mathématiques
Téléport 2 - BP 30179
Boulevard Marie et Pierre Curie
86962 Futuroscope Chasseneuil Cedex
France
Marc.van-Leeuwen@math.univ-poitiers.fr

Research

Computing Kazhdan-Lusztig-Vogan polynomials for split E_8

In 2007, Lie group theorists suddenly became the center of the media's attention when it was announced that the structure of the most complicated of the exceptional simple Lie groups, E_8, had finally been mapped. The precise result was in fact much more complicated, and might appeal less to the media: after intensive computer calculations, the Kazhdan-Lusztig-Vogan polynomials for the split real Lie group of type E_8 had been determined completely. The tale behind this computer algebra project is told by Marc van Leeuwen, professor at the Université de Poitiers, and one of the contributors to the solution of this problem.

On 19 March 2007, the project *Atlas of Lie groups and representations* suddenly found itself at the centre of a media blitz surrounding the announcement of the computation of the Kazhdan-Lusztig-Vogan polynomials for the split real Lie group of type E_8, which produced many gigabytes of data that could 'cover Manhattan'. While there had been a deliberate decision by the American Institute of Mathematics to organize publicity around a result obtained some months before, which would otherwise certainly not have attracted much attention outside the project, the sudden interest the announcement generated surprised all those involved. As would be the case for most subjects in mathematical research, it was impossible to explain to a general public what exactly had been done or why this was of such interest; we had to accept reading gross descriptions, such as that we had succeeded in finally 'decoding' E_8. One can only conclude that there must be many who, without knowing the details or seeking to, love mathematics in general and E_8 in particular. David Vogan has explained in the *Notices* of the American Mathematical Society the broader representation theoretic context in which the computations of the Atlas project take place and the one concerning E_8 in particular. In this article I will pursue a more modest goal of giving some indications of the nature of the computation and the mathematical objects involved.

Computational Lie theory and 'Atlas'

In Lie theory one studies groups that are also differentiable manifolds, a subject that at first sight seems not to offer the kind of possibilities for exact algebraic computation that exist for instance for finite groups. Yet the structure and representation theory of (reductive) Lie groups mainly employs derived combinatorial structures such as root systems rather than elements of the groups themselves and this in fact makes the subject very suited to a computational approach. Thus when sufficiently powerful computers became generally available around the 1980s, programs were developed for performing the essentially combinatorial computations occurring in this theory; a notable example is the program **LiE**, developed around 1990 in a project headed by Arjeh Cohen at the CWI in Amsterdam, which computes with representations of complex reductive Lie groups and which is still in use today. Kazhdan-Lusztig polynomials for Coxeter groups, which were first defined around 1980, are also of great importance in this theory, but although they are given by elementary recursion relations, their computation poses unique computational challenges that are best handled by a dedicated program. The most powerful such program *Coxeter* was written, and continually improved, during the 1990s by Fokko du Cloux, a Dutch mathematician living in Lyon.

In 2002, at a workshop on computational Lie theory held in Montreal, the first plans for a project that was to be named *Atlas of Lie groups and representations* were made by Jeff Adams, Dan Barbasch, Fokko du Cloux, John Stembridge, Peter Trapa and David Vogan. Their purpose was to apply computational methods also to the more complicated theory of real Lie groups. In particular the notion of Kazhdan-Lusztig polynomials can be generalised to this setting, although the 'parameter set' on which it depends is a considerably more complicated combinatorial structure than the Weyl group that is used in the case of complex groups. I became involved with this project when it was

already under way in 2003 through an invitation by Fokko (based on my experience in the LiE project) to participate in the considerable programming effort that this new project would require. After finishing his *Coxeter* program in early 2004, Fokko started working on this new program *atlas*. By the end of 2005 he had implemented all the necessary concepts from the structure theory of real reductive Lie groups and adapted the Kazhdan-Lusztig algorithms to this setting (while I was only starting to get to grips with the C++ programming language and some of the code he had written). At that point the tables of Kazhdan-Lusztig-Vogan (KLV) polynomials could be computed for all cases of interest, except for the big block of the split real form of E_8.

Classical and exceptional groups

To understand the special place of E_8 in computational Lie theory, let me say a few words about the classification of reductive Lie groups, simplifying somewhat to focus on the essential points. The first classification is that of *complex* reductive groups. These have a 'semisimple part' and a 'central torus'; the possible presence of the latter serves mostly to allow important examples like $\mathbf{GL}(n,\mathbf{C})$ (and to complicate certain algorithms) but is otherwise of minor importance. The semisimple part is essentially determined by its 'root system', which is a finite set of vectors in a finite (and relatively low) dimensional vector space, and it satisfies a number of symmetry conditions. These conditions are so restrictive that root systems can be completely classified, which was done by Killing and Cartan at the end of the 19th century. They are a product of 'simple' factors, each of which is either a member of one of four infinite families $(A_n)_{n\geq 1}$, $(B_n)_{n\geq 2}$, $(C_n)_{n\geq 3}$ and $(D_n)_{n\geq 4}$, or one of the five types labelled G_2, F_4, E_6, E_7 and E_8. The members of the infinite families occur as root systems of certain 'classical' Lie groups such as the special linear and symplectic groups; the remaining ones correspond to 'exceptional' simple Lie groups.

The classical root systems have a very regular description valid in all possible dimensions; for instance, the root system of type D_n can be realized as the set of all vectors of length $\sqrt{2}$ in the sublattice $\mathbf{Z}^n$ of $\mathbf{R}^n$ equipped with the standard scalar product (the cases D_2 and D_3 are omitted from the classification only because they coincide with certain other root systems). The exceptional root systems, however, depend on particular circumstances arising only in specific dimensions. For instance, the given description realizes the 24-element root system of type D_4 by vectors like $(1,0,-1,0)$, but a similar copy of this system scaled by a factor $\sqrt{2}$ can be found by taking all vectors of length 2 in $\mathbf{Z}^4$, like $(0,2,0,0)$ or $(-1,1,-1,-1)$; the union of these two systems forms the 48-element root system of type F_4. The systems of type G_2 and E_8 can be similarly formed due to singular circumstances, while those of type E_6 and E_7 are most easily described as root subsystems of the type E_8 system. The latter system has the following description. Starting with the usual 112-element root system of type D_8 contained in $\mathbf{Z}^8$, one adds all vectors in the lattice $(\frac{1}{2}\mathbf{Z})^8$ that (also) have length $\sqrt{2}$ and an even sum of coordinates, like $(\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2})$; this adds 128 vectors to make a total of 240 vectors for the type E_8 root system. This collection of vectors has an exceptionally high degree of symmetry (in which the members of the original D_8 subsystem are in no way distinguished from the others), as witnessed by the fact that for each of the 240 roots, the reflection in the hyperplane orthogonal to it stabilises the set of all roots; these reflections generate a finite group of order 696729600, the Weyl group.

In computational Lie theory many questions can be broken down according to the simple factors, so most attention is on handling those cases. Among them, the exceptional types have a special place for several reasons. First, as their number is finite, one may hope to completely solve certain questions for exceptional types by computational means. On the other hand the regularity of classical types makes it possible that answers to some questions can be given by an abstract (combinatorial) description, like the Littlewood–Richardson rule that describes tensor product decompositions in all types A_n. Also, exceptional groups, and in particular E_8, appear to have a more dense and complicated structure than classical ones, making computational problems

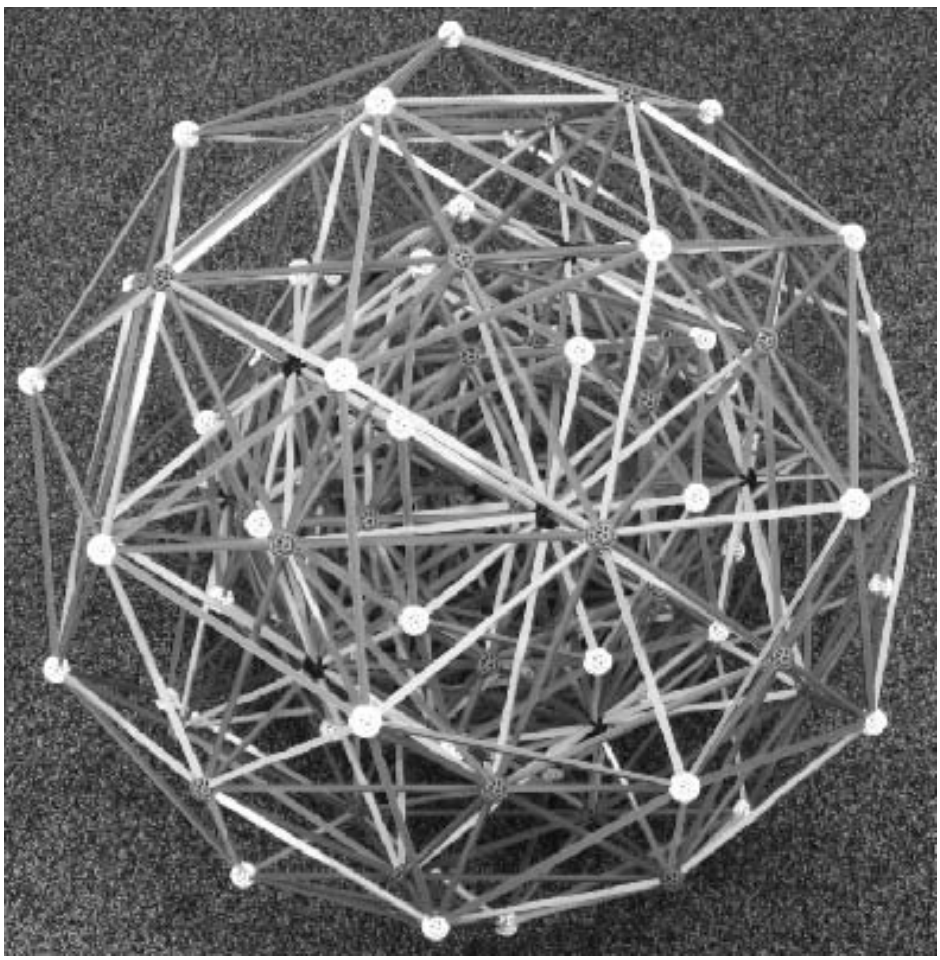
Members of the Atlas project; the fourth person from the left is Fokko du Cloux.

more challenging for them; this was in particular the case for the Kazhdan-Lusztig calculation. Thus E_8 serves as a 'gold standard': to judge the effectiveness of the implementation of a general algorithm, one looks to how it performs for E_8.

In the Atlas project the interest is in real Lie groups, which means that a 'real form' needs to be specified in a complex group. In type E_8 there are three inequivalent real forms. The mentioned parameter set for KLV polynomials, a combinatorial structure called a 'block', depends in a complicated way on the real form and on a real form in the dual complex group. Much of the code of the *atlas* program serves to construct blocks. The complex group E_8 happens to be its own dual, so there are 9 possible combinations of real form and dual real form; the *blocksizes* command of *atlas* tells how big the corresponding blocks are (some are empty):

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 3150 & 73410 \\ 1 & 73410 & 453060 \end{pmatrix}$$

So the first real form has just one block, with one element (this happens for the *compact* real form in every type), the second has two substantially larger blocks and the last block of the third (split) real form is referred to as the 'big block'. Since KLV polynomials depend on *two* parameters ranging over a block, the particular interest in the size of the big block can be imagined, a size that was not known before the *atlas* program was written. It turns out to be quite small compared to the Weyl group; but still, 453060^2 is approximately 2.05×10^{11}.



Source: http://upload.wikimedia.org/wikipedia/en/2/2b/E8.roots.zome.jpg

Figure 1 David A. Richter's *Zome* three dimensional model of the rootsystem of the E_8. See his website for building instructions and many other interesting models.

The challenge of the big block of E_8

Block elements correspond to classes of irreducible representations of the real Lie group considered, and KLV polynomials provide intricate structural information that applies to this class. For instance, the irreducible representations of a *compact* group, all finite dimensional, form a single class. The existence of a single KLV polynomial for them (in fact the constant polynomial 1) means that they are all governed by a single formula: Weyl's character formula (on which much of **LiE**'s computations are based).

A KLV polynomial is an element of $\mathbf{Z}[q]$ whose coefficients are known to be non-negative. The theory provides as the only means of computing them a collection of relations, expressed in terms of the combinatorial structure of the block, which can be used as recursion relations to successively compute these polynomials for all pairs of parameters ranging over a given block. This collection of polynomials is naturally thought of as a matrix, which is lower triangular, and recursion relations then refer to entries both in rows above and in columns to the right of the current entry. The nature of these relations makes it pointless to try to compute individual KLV polynomials or to compute them without storing the computed values for later retrieval. Thus *atlas* basically just fills this matrix of polynomials according to the recursion relations and then writes all results to a file.

As long as available memory suffices to store all the results, this computation is quite fast; this explains why all interesting cases other than the big block for E_8 could be handled fairly easily once the program was written. Even computing the KLV polynomials for the block of size 73410 for the middle real form of E_8 takes less than 10 minutes on a modern PC, provided it is equipped with at least 2 gigabytes (GB) of memory. This particular case can be handled because not *all* entries in the triangular part of the matrix are stored; that would amount to some 2.7×10^9 entries, certainly requiring much more than 10 GB. Any entry that some recursion relation simply equates to a previously computed one is not stored (it will be computed on the fly should its value be needed), nor is any entry that turns out to be zero. In the mentioned case this means that only 63 million entries, which amounts to 2.5% of the triangular region, are actually stored.

However, when memory runs out the situation deteriorates rapidly. By using virtual memory, one can avoid simply having to abandon the computation: part of the storage may be written to disk (to so-called swap space) to create space for new values. But although the operating system attempts to keep in memory the most recently used values, it turns out that this set shifts constantly so that quite soon the computation starts spending nearly all of its time swapping (exchanging values between memory and disk). This is what happened for the big block of E_8: even with several gigabytes of memory

and practically unlimited swap space, *atlas* was spending weeks just exercising the disk, while not even coming near to complete computation of the matrix of polynomials.

Meanwhile, we were shocked to learn that Fokko had been diagnosed with the neurodegenerative disease ALS, just around the time he finished programming the KLV algorithm; within months he lost the use of his hands and he died within a year. Though unable to program himself, he continued working on the Atlas project throughout that year, helping David Vogan, Jeff Adams and myself to get to grips with his program and to improve it further. Actually being able to do the big block of E_8 was not the primary concern for Fokko, as he told us that in a few year's time we would all have computers with enough memory to do it. He might very well turn out to have been right but in the end we did not have as much patience as he did.

Cracking E_8

We had contacted William Stein of Washington State University, who gave us permission to run *atlas* on a machine *sage* that had 64 GB of memory. However, it seemed that we would need a machine with at least 150 GB of memory to do the E_8 computation. While we considered buying an even larger computer, an idea occurred that triggered an attempt to adapt *atlas* so that it could complete the E_8 computation on *sage*. Two characteristics distinguishing the big block of E_8 from simpler cases are that the number of *distinct* polynomials in the matrix grows relatively fast (many polynomials occur extremely often so *atlas* stores only one copy of each) and that some polynomials require rather large coefficients (larger than $2^{16} = 65536$), which meant we needed to reserve 4 bytes for each coefficient. Combined, this means that a lot of storage is needed just to record all the polynomials occurring: it involves more than 13 billion 4-byte coefficients. The idea was that the form of the recursion relations allowed the computation to be performed multiple times with modular coefficients, each of which can be stored in a single byte, and that the Chinese remainder theorem could be used afterwards to reconstruct the actual integer coefficients.

As the *atlas* program is well-structured, it was quite easy to modify it so that storage for polynomial coefficients was reduced to one byte and that all arithmetic on them was replaced by modular arithmetic. This modified program was ready in early December 2007, just a few days after the idea of using modular arithmetic had been adopted (and less than a month after Fokko's death). However, this modification, even though it divided the storage requirements for coefficients by 4, did not suffice to be able to do the big block even on *sage*, as other important factors of storage requirement were unaffected, i.e. the matrix telling which polynomial occurs where, the data structure used to allow each polynomial to be represented just once and the general overhead of data structures. Normally one trusts the libraries used to provide convenient data structures and does not bother to fiddle with details of bits and bytes; however, with data types that occur billions of times and memory at a premium, it pays to push for improvement in these areas. In fact it turned out that much could be gained with fairly straightforward techniques like replacing 8-byte pointers with 4-byte identification numbers, using a hash table rather than a search tree and gathering all polynomial coefficients in a single array rather than in a billion different ones. I had been familiar with such methods since a long time, having studied programs (like TEX) written by Donald Knuth at a time when computer memory was less abundant than it is today. In the end, the memory requirement for the big block was brought down to about 55 GB, making it feasible to try it on *sage*.

Everything, including some small utility programs to perform lifting of modular coefficients, was more or less ready to run at the start of the Christmas holidays. The computations were actually run in Seattle via remote control by David Vogan, from MIT or whatever other place with Internet connection (like an airport lounge) he happened to be at, and with other Atlas members 'looking over his shoulder' through VNC windows. I will not repeat the beautiful description in David's article of our varying fortunes as the computations progressed. Suffice it to say that every first attempt at a part of the computation failed, either because of a programming error that previous testing had failed to unveil or due to one of many system crashes (which turned out to be caused by independent processes on *sage*). Nevertheless, with obstinacy we managed to push on to the finish line on 8 January 2008.

My personal recollection of this adventure is one of some short interruptions of the holidays spent at my home in Poitiers with my two teenage children, in which I discussed software problems with David by email (he actually found and repaired most of them himself), and of a final weekend back in the Netherlands where, after having returned the children to their home, I spent an evening at my parents' pondering over a last bug that David had signalled. Early the next morning, having found that an overnight trial run had failed to reproduce any error, I wrote an email back to give David the go-ahead for another try (because the bug had accidentally been fixed already!), while my father was worrying that I was going to miss my train back to France (which I didn't). For once, computers needed more time than trains: I was already back to work the next day before the final run of the Chinese remainder program completed.

References

1 David Vogan, "The character table for E_8" *Notices Amer. Math. Soc.* **54** (2007), no. 9, 1122–1134.

2 http://www.aimath.org/E8, beschrijving door het American Institute of Mathematics

3 http://www.liegroups.org, de website van het Atlas project