Research project description

My research project culminated in a thesis entitled *Conjugacy and centralisers in Thompson's group T*. In this short, nontechnical essay I'll describe what this title actually means, the problems I encountered and how I went about solving them.

What's a group? Why do people care about them?

A group is a mathematical gadget which describes the types of symmetry an object possesses. One good example comes from 2D shapes. A square can be rotated about itself only through 90°, 180°, 270° or 360° angles. This is different from a circle, which can be rotated through absolutely any angle from 0° to 360°! Mathematically, we recognise this by assigning two different *symmetry groups* to the square and the circle.

There are lots of things which can reasonably be called a symmetry. Other geometric examples include translation and reflections (perhaps even in 3D), but there's a whole zoo out there of things that mathematicians call 'symmetry'. As far as maths is concerned, the idea is very abstract: all we need is a consistent way to *combine* symmetries. Thinking back to the square, we could combine a 90° rotation with a 180° rotation to get a 270° rotation.

Why bother studying symmetry and groups? I think of it chiefly as a time-saver. If we want to study an object X in detail, there may be lots of possibilities or cases to worry about. If X has lots of symmetry, there's a good chance that many of those cases will overlap, being duplicates of each other. Then we only need to reason about the *essential* cases: we know that all the rest will follow on, thanks to our understanding of symmetry. So we can save time by reducing to the important bits of an argument. My favourite example of this is the article "There Is No 16-Clue Sudoku".[1] They use symmetry arguments to shorten a computer search for solvable Sudoku grids by a factor of 1.2 trillion!

Sometimes though, the symmetries of an object—all bundled together in one of these 'groups'—are worth studying in their own right. One example comes from chemistry, where the symmetries of a molecule or a crystal structure can give clues as to its chemical properties. Another more abstract mathematical example uses symmetries as a tool. Imagine it's hard to distinguish between two mysterious objects. If we could show that the objects have different symmetries (different symmetry groups), we can conclude that our objects must be different.

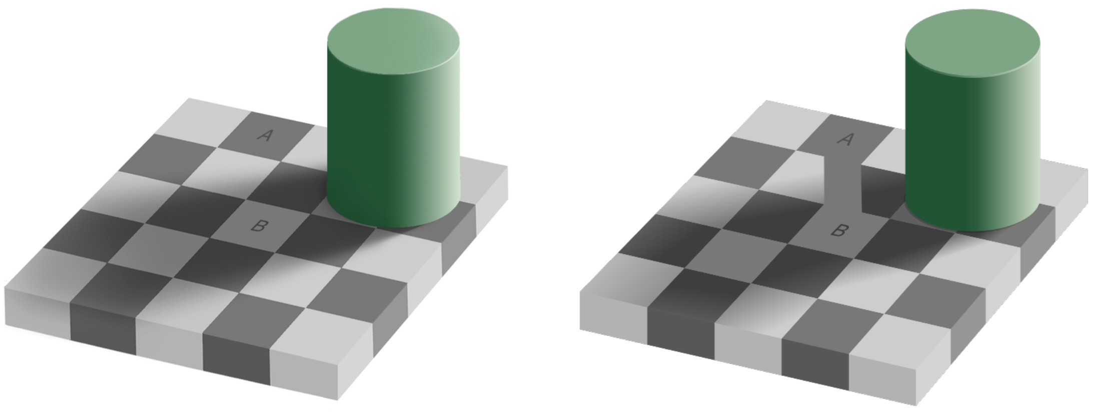
One final application of group theory is to cryptography—encrypting and decrypting communications. Modern cryptography relies on hard problems which become easy when we possess some extra secret information. Very roughly speaking, a computer would have to take ages to solve a hard problem, but hardly any time to solve an easy problem. Group theory is a source of computational problems whose difficulty can be *quantified*. This means that we have a good numerical estimate of how long it'll take the computer to solve the problem, and so we can get an idea of how hard these problems are. The hope is that in studying groups, we might find new problems which are suitable for cryptography. For example, the problem at the heart of the Diffie–Hellman key exchange protocol—discrete logs—can be formulated for any group.

[1] Gary McGuire, Bastian Tugemann, Gilles Civario. *Experimental Mathematics* 2014 23:2, pp 190–217. See their section 3 in particular.

What does the rest of your jargon mean?

My thesis studies one particular group called 'Thompson's group T'. This group is a collection of unusual symmetries of a circle, wherein some parts of the circle are stretched and others are squashed. (Think of pulling apart two close points on an elastic band.) It's best known to other group theorists as a source of unusual examples. Indeed, T and its relatives are infamous groups for which many reasonable questions are hard to tackle.

'Conjugacy' is the idea that two different symmetries can look like the same when viewed in the right way. If so, the two different symmetries are called *conjugate*. It's a bit like an optical illusion: two objects that look different at first glance are revealed to be the same—from the right point of view. In the images below, square A looks darker than square B. However, both squares are exactly the same colour; our brains are tricked into thinking otherwise thanks to the shadow over B.



Images: Wikipedia article for 'Optical illusion', licensed for free use.

My job was to solve the conjugacy search problem in Thompson's group T. This means I had to design an algorithm to decide if two given symmetries were conjugate. If so, the algorithm had to explain why—which is harder. Put differently, the algorithm had to produce *evidence* that the two symmetries were conjugate. To make an analogy with the coloured squares above, the algorithm would have to perform the equivalent of adding the grey strip between A and B in the right-hand image. On top of that, I needed to produce a formal, mathematical proof that the algorithm does what it claims to do.

The last bit of jargon is a 'centraliser'. I mentioned earlier that in a group we have a way to combine symmetries. We use multiplication to denote this, so that combining a and b results in $a \times b$. Crucially, the order of combining matters: it may be that $a \times b$ is not the same as $b \times a$! Centralisers are another group theory gadget for measuring how often this occurs. When centralisers are large, we find $a \times b = b \times a$ is true most of the time. Conversely, small centralisers mean that $a \times b = b \times a$ only very rarely; in this situation the group is more complicated. I went on to study these gadgets for Thompson's group T, because centralisers can be used to 'count' the number of ways in which two symmetries are conjugate. Put differently: centralisers allow us to find *all* pieces of evidence that two symmetries are conjugate, instead of just one or two pieces.

What did you do, and how?

The Thompson groups are fascinating, but are a pain to work with by hand. Each element (i.e. each symmetry) can be represented in an infinite number of ways, and it's tricky to choose the right representation for the job. Worse still, calculating with elements is tedious and error-prone. To get started, I wrote a series of Python scripts to help automate these calculations, giving me the chance to learn about the groups while doing so.

My initial goal was to implement an algorithm described in a draft paper. I did so in Python, because I was familiar with the language, and because it's well-suited for rapid prototyping. To start, I established a way of representing group elements in memory. I made use of object-oriented programming to keep things organised. As the software grew more complex, I used tools for documentation, version control and testing. The tests were quick to run, yet functional enough to catch bugs before I discovered them!

With data structures in hand, I began to implement the algorithm as described in the draft paper. Surprisingly, the paper had an error, which boiled down to a gap in an argument going back to the 70s. Fortunately, this wasn't a complete disaster: I was able to work around the gap by storing some extra data and by adding a few extra steps to the algorithm. My contribution was recorded in the draft article, later published in 2016.

Officially, my project was to solve the 'simultaneous conjugacy problem' in Thompson's group V. This is a harder problem than just the conjugacy problem, and V is larger and more complicated than T. I spent a lot of time reading the literature on this, but eventually I found myself stuck and getting nowhere fast. One difficulty was understanding how different V-centralisers interacted—almost anything could happen here. Another difficulty came from the difficulty of working with the non-unique data structures that describe conjugacy in V. I wrote some experimental scripts to develop a heuristic approach, but these weren't robust enough to make a formal algorithm.

Instead, I re-read the thesis of F. Matucci, whose work includes the T Thompson group mentioned earlier. To adapt Matucci's algorithms to work in the T group, I had to learn new mathematics. This enabled me to consistently describe and work with points on a circle, even when that circle is being stretched and squashed. With some effort, a case-by-case analysis and a number of small details to be checked, I was able to extend and adapt Matucci's techniques to solve the conjugacy problem in T.

In the group theoretical world, conjugacy and centralisers go hand-in-hand. Mattuci had found the 'ingredients' to describe centralisers in T, but he was not able to provide the recipe for combining them to form a fully-fledged centraliser. I found an example element whose recipe I could deduce, and suspected this was part of a larger pattern. I began another case-by-case analysis to try and describe this pattern, starting by working out what parameters influenced the centraliser I was seeking.

Alongside my work I was continuing to develop and maintain my Python package. Here my software was invaluable. I was able to quickly validate or refine my approach, by having the computer perform concrete calculations in order to test my theoretical calculations. This gave me the confidence to continue, and ultimately conclude my thesis with a classification of certain centralisers in T.

The end product

My initial software and fix to the 'gap' forms part of Barker, Duncan and Robertson, 2016; see my CV. The software I wrote is available online,[2] and my submitted thesis is unofficially available online too.[3] I am working on presenting the thesis's main results in two standalone articles.

[2]`https://github.com/DMRobertson/thompsons_v`

[3]`https://www.mas.ncl.ac.uk/~b0036119/thesis_submitted.pdf`