Mathematics 6310
The Todd–Coxeter procedure
Ken Brown, Cornell University, April 2011

The Todd–Coxeter procedure is a systematic way of trying to find the order (and often the structure) of a group given by generators and relations, provided the group is finite. More generally, we can try to find the index $|G : H|$ of a subgroup H of finite index, even if G itself is infinite. The problem of finding the order is the special case $H = \{1\}$. A good introductory reference is Neubüser [5] (supplemented by [1].) For a more detailed treatement and further references, see [4, Chapter 5]. In this handout we will confine ourselves to a brief overview and a simple example

We will describe the method in terms of Schreier graphs, so we begin by recalling what those are.

1. Schreier graphs

If σ is a permutation of a set X, you can picture σ by drawing a directed graph with X as vertex set and with an edge from x to $\sigma(x)$ for each $x \in X$. Thus each vertex has exactly one edge coming in and one going out. Note that the graph breaks up into connected components, one for each cycle in the cycle decomposition of σ.

If you want to picture two or more permutations of X simultaneously, you can use a different color for each one of them. Then each vertex has exactly one edge of each color coming in and one going out.

If you want to picture a whole group of permutations, you don't have to draw them all; it's enough to draw the pictures of a set of generators of the group. You can then read off the action of any element of the group by writing it as a word in the generators and their inverses and following the corresponding path in the graph. Note that this works much better for right actions than for left actions, since you can follow the path step by step as you read the word from left to right.

2. The Todd–Coxeter procedure

Suppose G is a group given by generators and relations and H is a subgroup of finite index. The idea of the Todd–Coxeter procedure is to try to figure out what the Schreier graph of the (right) action of G on $H\backslash G$ looks like. What we do is make up names for cosets (typically 1,2,...) as we go along, and draw in arrows for the action of the generators as we discover them. The process stops when the graph is "complete". This means:

- Every vertex has exactly one edge of each color coming in and one going out.
- The graph shows that all the relations are satisfied.

I will illustrate the method with the following example

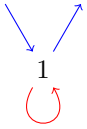
$$G := \langle\, a, b \,;\, a^3 = b^3 = (ab)^2 = 1 \,\rangle\,.$$

Let $H := \langle a \rangle \leq G$, and consider the action of G on $X := H\backslash G$. We will use red arrows to denote the action of a and blue arrows to denote the action of b. When the Schreier graph is complete, every red-red-red path will have to be closed as a result of the relation $a^3 = 1$. Similarly, every blue-blue-blue path will have to be closed and every red-blue-red-blue path will have to be closed. So if we see at

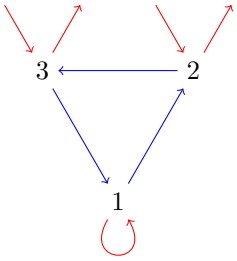
some stage a subgraph consisting of two consecutive arrows of the same color, then we can fill in a third arrow to close the triangle. Similarly, if we see a subgraph consisting of three sides of a red-blue-red-blue square, then we can fill in the fourth side.

We now try to construct the Schreier graph.

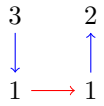
Step 1. Record the fact that there is an element of X (called "1") fixed by H. The two short blue arrows are reminders that we will eventually have to have a blue arrow coming in and a blue arrow coming out. But we don't yet have names for the vertices at the other ends.



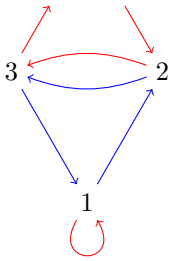
Step 2. We now check our three relations at the vertex 1. (i) The relation $a^3 = 1$ holds trivially (i.e., the red-red-red path starting at 1 is closed). (ii) In order to check the relation $b^3 = 1$, we introduce the names $2 := 1 \cdot b$ and $3 := 2 \cdot b$. The relation then tells us that we can close the blue triangle, yielding



Again we have drawn short arrows at the two new vertices as reminders of what is missing. (iii) Finally, we check the relation $abab = 1$ at vertex 1, i.e., we look for a red-blue-red-blue square starting at 1. Three sides of that square are already present:

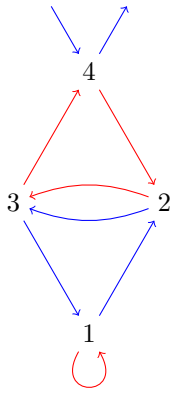


As noted above, we can fill in the square, i.e., we can deduce that $2 \cdot a = 3$. So our graph now looks as follows:



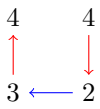
It is complete at vertex 1.

Step 3. We now move on to vertex 2 and check the three relations. (i) To check $a^3 = 1$, we define $4 := 3 \cdot a$ and then close the red triangle. This yields:

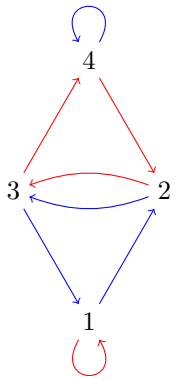


(ii) The relation $b^3 = 1$ already holds at vertex 2. (iii) The relation $abab = 1$ also holds. So the graph is complete at vertex 2.

Step 4. At vertex 3, the relations (i) and (ii) already hold. But in checking relation (iii), we see that we have three sides of the red-blue-red-blue square:



We fill in the fourth side and deduce that $4 \cdot b = 4$. Our graph is now



It is complete at 3.

Step 5. Finally, we move to vertex 4 and check that all three relations hold. The graph is complete.

The conclusion is that $|G : H| = 4$. [Exercise: Justify this assertion rigorously.] Since $|H|$ is 1 or 3, it follows that $|G|$ is 4 or 12. But we get much more. Namely, the Schreier graph lets us exhibit a homomorphism $G \to A_4$, taking the generators a and b to the red and blue permutations. It is easy to check that these two permutations generate A_4, so the homomorphism is surjective. Thus $|G| = 12$, and in fact $G \cong A_4$.

Remark. This example was simpler than some, in that all the names we introduced (1, 2,...) turned out to represent different cosets. A more common situation is that some some cosets with different names are discovered to be identical, at which point one has to redraw the graph (which becomes simpler). This situation is known in the literature as a *coincidence*. Coincidences are inevitable in general. [Think about presentations of the trivial group, for instance.]

If you want to try some examples on your own, here are a few where the Todd–Coxeter method reveals the structure fairly easily:

- $\langle\, a, b, c \,;\, bab^{-1} = a^2,\ cbc^{-1} = b^2,\ aca^{-1} = c^2 \,\rangle$ is the trivial group.
- $\langle\, a, b \,;\, ab^2a^{-1} = b^3,\ ba^2b^{-1} = a^3 \,\rangle$ is also the trivial group.
- $\langle\, a, b \,;\, a^3 = b^2 = (ab)^4 = 1 \,\rangle \cong S_4$.
- $\langle\, a, b \,;\, a^3 = b^5 = (ab)^2 = 1 \,\rangle \cong A_5$.
- $\langle\, a, b \,;\, a^2 = b^2 = (ab)^2 \,\rangle \cong Q_8$.

The last one is perhaps the most surprising. Note that there's no "= 1" at the end of the relations. So it's not even obvious that a and b have finite order. But somehow the relations force them to have order 4.

3. Strategy

It is clear that we have many choices in carrying out the Todd–Coxeter procedure. We can make definitions of new vertices at any time and we can try to check any relator at any time at any vertex. The strategy we followed above is often called the *HLT* method (after Haselgrove, Leech, and Trotter). In this method we consider the vertices in numerical order, and we make sure that the graph is complete at each vertex before moving on to the next. The references cited above discuss this and other possible strategies in detail.

There is a variant of HLT, called *HLT plus lookahead*, which can be useful if we want to keep the Schreier graph from getting unnecessarily big. Suppose we are working at vertex v and the graph that we've constructed so far is very large. Instead of checking relators at v and making new definitions so that the graph is complete at v, we pause and look ahead at later vertices in the hope of discovering coincidences that will shrink the graph. To this end, we check all relators at all vertices (but without making new definitions) and take account of any deductions or coincidences that we can discover. We then go back to vertex v and continue the HLT procedure with the new simplified graph.

A third method in common use, called the *Felsch* method, goes as follows. We still proceed from vertex to vertex, but we don't necessarily check the relators at the current vertex. Instead, we make new definitions (if necessary) so that the given vertex has all the required incoming and outgoing arrows. After each definition, we check all relators at *all* vertices (without making further definitions) that could possibly be affected by the definition we've just made.

4. Comments

It can be shown that the Todd–Coxeter procedure will always terminate if the index of H in G is finite. Considering the case $H = \{1\}$, it follows that the Todd–Coxeter procedure can always be used (in principle) to determine the order of a finite group given by generators and relations. Why doesn't this contradict the

assertion I made in class that it is algorithmically impossible to discover anything interesting about a group from a presentation?

The answer is that we can't determine from the presentation whether or not the Todd–Coxeter procedure will terminate, so we can't even decide whether or not the group is finite. But if we happen to know somehow that the group is finite, then the Todd–Coxeter procedure does indeed determine its order (in principle).

I say "in principle" because there are also the practical issues that we can't predict how much time it will take or how much memory it will require.

In spite of these limitations, Todd–Coxeter seems to work pretty well in practice. It's built into many computer algebra systems, in case you want to try some examples that are too big to do by hand. See Section 6 below for more information about available software.

5. Coset tables

Most of the literature describes the Todd–Coxeter procedure in terms of *coset tables* instead of Schreier graphs. A coset table simply lists the cosets and displays the action of the generators and their inverses. In other words, it lists the vertices of the Schreier graph and displays the endpoints of all outgoing and incoming arrows. For example, the complete coset table for the example treated in Section 2 is

	a	A	b	B
1	1	1	2	3
2	3	4	3	1
3	4	2	1	2
4	2	3	4	4

Each row corresponds to a coset, and each column corresponds to a generator or its inverse, with the convention that uppercase letters denote the inverses of the generators. Row 2, for example, shows that $2 \cdot a = 3$, $2 \cdot a^{-1} = 4$, $2 \cdot b = 3$, and $2 \cdot b^{-1} = 1$. This is the same information that we would read off from the Schreier graph by looking at the arrows entering and leaving vertex 2.

During the course of the Todd–Coxeter procedure, some of the entries in the table are not known. One common convention is to enter 0 in these places. For example, the coset table at the end of Step 2 in Section 2 is

	a	A	b	B
1	1	1	2	3
2	3	0	3	1
3	0	2	1	2

At this stage coset 4 has not yet been defined.

As I mentioned in the remark in Section 2, coincidences may occur when one carries out the Todd–Coxeter procedure. Thus rows may "die", leaving gaps in the numbering. It is customary to remove the dead rows at the end of the process and to renumber the cosets so that there are no gaps. This is often called *compression* in the literature.

Finally, it is sometimes convenient to renumber the vertices at the end so that the table is in a certain "standard" form. The details need not concern us, but I mention it in case you see references to it in software that you use. (In our

example, the table happened to come out in standard form, so no renumbering was necessary.)

6. Software

Among the many computer algebra systems that incorporate the Todd–Coxeter procedure, my favorite is GAP [3]. It can carry out the procedure in a variety of ways. One of them is in a package called ITC [2] ("interactive Todd–Coxeter") which lets you control the process step by step and experiment with different strategies. Start by looking at the ITC manual if you want to know more.

I have also written my own self-contained Todd–Coxeter program for use on small examples. It is written in `C++` and is not efficient in terms of speed and memory use. My main goal in writing it was to make it simple and straightforward, so that you can read the code and understand how it relates to the algorithms that were described informally in Section 3 above. If you want to try it, see `http://www.math.cornell.edu/~kbrown/toddcox/`. You can build it yourself or download binaries that will run on the Math Department's Linux system or on Windows.

References

[1] Colin M. Campbell, George Havas, and Edmund F. Robertson. "Addendum to: "An elementary introduction to coset table methods in computational group theory"". In: *Groups—St. Andrews 1981*. Vol. 71. London Math. Soc. Lecture Note Ser. Cambridge: Cambridge Univ. Press, 2007, pp. 361–364.

[2] Volkmar Felsch, Ludger Hippe, and Joachim Neubüser. *GAP package ITC: Interactive Todd–Coxeter*. 2004. URL: `http://www.gap-system.org/Packages/itc.html`.

[3] *GAP – Groups, Algorithms, and Programming, Version 4.4.12*. The GAP Group. 2008. URL: `http://www.gap-system.org`.

[4] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005, pp. xvi+514. ISBN: 1-58488-372-3. DOI: `10.1201/9781420035216`. URL: `http://dx.doi.org/10.1201/9781420035216`.

[5] Joachim Neubüser. "An elementary introduction to coset table methods in computational group theory". In: *Groups—St. Andrews 1981*. Vol. 71. London Math. Soc. Lecture Note Ser. Cambridge: Cambridge Univ. Press, 2007, pp. 1–45.