Algorithms for Computations in Finite Groups

Shraddha Ramesh sramesh@sfu.ca
Michael Monagan mmonagan@cecm.sfu.ca

NSERC CRSNG

Introduction

The *group* package in Maple already has a representation for permutation groups, *permgroup*, which requires users to specify the generators and degree of the permutation group. We are building a new package called *FiniteGroups*. We have implemented new forms of representing a group G, *PermGroup* and *MatrixGroup*, where the user has the option of defining a number of properties of G, including: group multiplication operation, procedure to find the inverse of an element, identity, and characteristic.

Procedures for group computations have been implemented, such as: generating the elements in a group, drawing the Cayley table of a group, and generating a random element in a group, all of which require the user to simply provide a set of generators of the group, as well as the degree in the case of a permutation group. These are input in the form of *permgroup*, *PermGroup* or *MatrixGroup*.

Generating Elements of a Group

Given a set of generators for a group G, *GenerateGroup* will generate the elements in G using Dimino's Algorithm, and display the list of elements.

```
> S3 := permgroup( 3, { [[1,2]], [[1,3]] } ):
> GenerateGroup(S3);
```

[[], [[1, 2]], [[1, 3]], [[1, 2, 3]], [[1, 3, 2]], [[2, 3]]]

```
> Quaternion := MatrixGroup( { Matrix([[I,0],[0,-I]]),
Matrix([[0,1],[-1,0]]) } ):
> GenerateGroup(Quaternion);
```

$$\left[\begin{bmatrix}1&0\\0&1\end{bmatrix},\begin{bmatrix}I&0\\0&-I\end{bmatrix},\begin{bmatrix}-1&0\\0&-1\end{bmatrix},\begin{bmatrix}-I&0\\0&I\end{bmatrix},\begin{bmatrix}0&1\\-1&0\end{bmatrix},\begin{bmatrix}0&-I\\-I&0\end{bmatrix},\begin{bmatrix}0&-1\\1&0\end{bmatrix},\begin{bmatrix}0&I\\I&0\end{bmatrix}\right]$$

Dimino's Algorithm can quickly generate groups of orders of up to hundreds of thousands. The number of group multiplications that Dimino's Algorithm performs is related to the order of the group: $m \approx n^{1.012}$ for the highest order groups tested, where m is the number of multiplications and n is the order of the group.

Order of Group	Number of Multiplications	Time (seconds)
6	12	0.001
24	40	0.002
60	100	0.004
720	1080	0.056
5040	6048	0.260
20160	22848	2.202
40320	45696	3.546
362880	423360	33.945

Table 1: The number of multiplications performed and the time taken by Dimino's Algorithm for permutation groups of various orders.

Group Representation

We show an example of the representation for a matrix group. If the user does not supply group information, defaults are used.

```
> Quaternion := MatrixGroup( { Matrix([[I,0],[0,-I]]),
Matrix([[0,1],[-1,0]]) } ):
```

Quaternion := MatrixGroup(*generators* = $\left\{\begin{bmatrix}I&0\\0&-I\end{bmatrix},\begin{bmatrix}0&1\\-1&0\end{bmatrix}\right\}$,

characteristic = 0, *multiply* = **proc**(*A::Matrix, B::Matrix*) '.'(*A, B*) **end proc**,

inverse = **proc**(*A::Matrix*) 1/*A* **end proc**, *identity* = $\begin{bmatrix}1&0\\0&1\end{bmatrix}$,

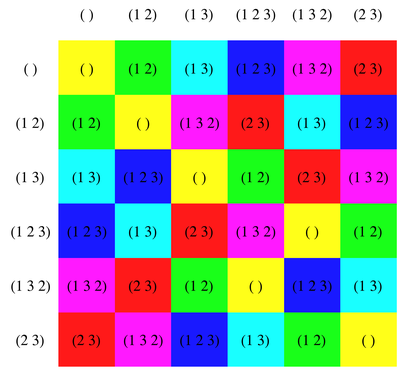
hashfunction = **proc**(*A::Matrix*) *convert*(*A, list*) **end proc**)

Cayley Table

An algorithm to display the Cayley table of a group has been implemented. Each element of the group is given a separate colour, and the user can specify whether or not the elements will be labelled, as well as the size of the labels.

We draw the Cayley table of S3, the symmetric group of degree 3.

```
>CayleyTable( S3, labels=true, labelsize=12 );
```



Orbit of an Element

The *group* package on Maple already has a function *orbit* that, given α and the generators of a permutation group G, returns the orbit of α in G. Denote the orbit of α by α^G. We have implemented a function *OrbitStabilizer* that outputs, along with each element $\beta \in \alpha^G$, an element $u_\beta \in G$ such that $\alpha^{u_\beta} = \beta$. That is, for each element $\beta \in \alpha^G$, we also give an element in G that takes α to β.

```
> G := permgroup( 6, { [[1,2],[3,5]], [[1,4]] } )
```

Compute the orbit of 2 in G using Maple's *orbit* function:

```
> orbit( G, 2 );
```

{1, 2, 4}

```
> OrbitStabilizer( G, 2 );
```

{{1, [[1,2],[3,5]]}, {2,[]}, {4, [[1,2,4],[3,5]]}}

Random Element

Given a set S of generators of a group G, *RandomElement* returns a random element in G. The algorithm used for the implementation is based on the *product replacement algorithm* found in Holt [1]. For a given group G, *RandomElement* first executes an initialization step, as follows:

An array A is initialized, containing the generators $s_1 ... s_k \in S$ repeated a number of times, r. We take $r = 5$. For example, if $S = \{s_1, s_2\}$ then A would be the following array:

$$[s_1\ s_2\ s_1\ s_2\ s_1\ s_2\ s_1\ s_2\ s_1\ s_2]$$

The initialization step then does the following *basic operation* a given number of times to move away from the original generating set: random distinct integers u and v are picked, with $1 \le u, v \le r \cdot k$. $A[u]$ is then replaced by one of the following, chosen randomly: $A[u] \cdot A[v]$, $A[u] \cdot A[v]^{-1}$, $A[v] \cdot A[u]$, or $A[v]^{-1} \cdot A[u]$. Note that the entries in A will still generate all the elements of G. We use a default value of 50 repetitions of the *basic operation*, but for large groups this may not be sufficient. Therefore the user has the option of inputting how many repetitions will take place.

After the initialization step, to generate a random element, *RandomElement* simply carries out the *basic operation* another time, and returns the new value of $A[u]$.

An example:

```
> RandElem := RandomElement(Quaternion):

> RandElem(), RandElem(), RandElem();
```

$$\begin{bmatrix}0&-1\\1&0\end{bmatrix},\ \begin{bmatrix}-I&0\\0&I\end{bmatrix},\ \begin{bmatrix}I&0\\0&-I\end{bmatrix}$$

References

[1] D.F. Holt, B. Eick, and E.A. O'Brien. *Handbook of Computational Group Theory.* Boca Raton: Chapman & Hall/CRC Press, 2005.

[2] D. Joyner. *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys.* 2007.