

Solutions for Workshop

Randomised Algorithms

Alice C. Niemeyer

1. See accompanying GAP-function `IsPurpleElement`. It takes as input a permutation g and a number n . If $g \notin S_n$ it raises an error; it returns `true` if $g \in S_n$ is purple and `false` else.
2. See accompanying GAP-function `ContainsAlternatingGroup`. It takes as input a permutation group and a number N . If in at most N random selections from the group it can prove that the group contains A_n it returns a list `[true, i]`, where i is the number of random selections required, and it returns `false` if it could not prove that the group contains A_n . It returns `false` if the group certainly does not contain A_n .
3. The following example commands run the algorithm 100 times for input group S_{1000} allowing 10000 random selections at most in each run. The code counts the number of failures in f and the accumulative number of random elements selected in s .

```
sn := SymmetricGroup( 1000 );
Sym( [ 1 .. 1000 ] )
gap> f := 0;; s := 0;;
gap> for i in [ 1.. 100 ] do
> r := ContainsAlternatingGroup(sn,10000);
> if r <> fail then
> s := s + r[2];
> else f := f + 1;fi;
> od;
gap> f;
0
gap> s;
1056
```

Hence in this example, the algorithm recognised that the group contains A_{1000} on average in 10.56 random selections. Now $\log_2(1000) = 9.965784$.

The following table shows the performance of the algorithm with input S_n for various values of n . It lists n in the first column, the average number of random selections required in the second column, and $\log_2(n)$ rounded to the first digit in the third.

| n | number | $\log_2(n)$ |
|-------|--------|-------------|
| 10000 | 15 | 13.3 |
| 1000 | 11 | 10.0 |
| 100 | 8 | 6.6 |
| 50 | 7 | 5.6 |
| 10 | 8 | 3.3 |

It can be seen from the table that the algorithm requires about $O(\log_2(n))$ random selections. It performs better for larger n than for smaller n . It is very clear that it uses far less than n random elements.

4. See accompanying GAP-function `FindNCycle`. It takes as input a group, a number n , and a number N . It chooses at most N random elements in the group and tries to find one whose order divides n . It returns `fail`, if it failed to find an element of order dividing n and a list $[g, i]$ otherwise, where g is an element of order dividing n and i is the number of random selections required to find it.
5. The following GAP-code ran the function `FindNCycle` for input S_{1000} a total of 100 times and recorded the accumulated number elements of order dividing n which were not n -cycles in f and the accumulated number of random selections in s . We allowed 100000 random selections at most to ensure that each run found an element of order dividing n .

```
sn := SymmetricGroup( 1000 );
Sym( [ 1 .. 1000 ] )
gap> n := 1000;;
gap> f := 0;; s := 0;;
gap> for i in [ 1 .. 100 ] do
> r := FindNCycle( sn, 1000, 100000);
> if r <> fail then
> c := CycleStructurePerm(r[1]);
> if IsBound( c[n-1]) then s := s + r[2];
else f := f + 1;
fi;
```

```
> od;  
gap> f;  
1  
gap> s;  
98983
```

Hence for S_{1000} an n -cycle was obtained on average in 990 random selections, which is pretty close to the predicted value of $n = 1000$. Only one of the permutations whose order divided 1000 was not n -cycles. Hence in this example, when `FindNCycle` returns permutation it has a very good chance of being an n -cycle. This matches the theory.