Discrete mathematics with R: introducing the permutations package

Robin K. S. Hankin

Abstract

Here I introduce the **permutations** package, for manipulating and displaying permutations of a finite set. I show how the package has been used to investigate the *megaminx* puzzle, and exhibit an 82-turn superflip.

Keywords: Permutations, megaminx, superflip.

1. Overview

Permutations of a finite set are important and interesting mathematical objects, having applications in combinatorics (Stanley 2011), group theory (Milne 2013), and various branches of recreational mathematics (Averbach and Chein 2000).

Open-source computer software for working with permutations includes the GAP suite of software, for which Sage is a popular front end (Joyner 2008). However, these systems are designed for the pure mathematician with a focus on formal algebraic properties of specified groups, such as homology and the identification of Sylow subgroups. The **permutations** package is offered as an R-centric suite of software to carry out computational manipulation of finite permutations.

2. The package in use

It is a common occurrence that a discrete set of objects is rearranged in some way. Mathematically, this is described by a bijection from the set of objects to itself.

For example, consider bijections from the set $[n] = \{1, 2, 3, \ldots, n\}$ to itself. There are $9! = 362880$ such bijections and, as a specific example we will consider $f\colon [9] \longrightarrow [9]$ defined by the following diagram:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 2 & 6 & 3 & 5 & 4 & 1 & 7 & 8 \end{pmatrix}$$

Thus $f(1) = 9$, $f(2) = 2$, and so on. The **permutations** package would represent $f(\cdot)$ by specifying the images of $1, 2, \ldots, 9$ in sequence:

```
> library(permutations)
> f <- as.word(c(9,2,6,3,5,4,1,7,8))
```

```
> options(print_word_as_cycle = FALSE)
> f
```

```
    {1} {2} {3} {4} {5} {6} {7} {8} {9}
[1] 9   .   6   3   .   4   1   7   8
```

The images of each set are shown below the set element which is itself indicated by curly brackets. Elements which map to themselves, viz 2 and 5, are shown by the print method[1] as a dot; this makes complicated permutations easier to view as one is frequently not interested in fixed elements.

In this example, f is known as a *word*: the R object specifies the images of the base set in order. Function $f(\cdot)$ is invertible, being a bijection. Its inverse may be found by inspection:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 2 & 4 & 6 & 5 & 3 & 8 & 9 & 1 \end{pmatrix}$$

The R idiom would be

```
> inverse(f)
```

```
    {1} {2} {3} {4} {5} {6} {7} {8} {9}
[1] 7   .   4   6   .   3   8   9   1
```

(see how the fixed elements remain fixed). Permutation inverses are carried out using the compact and efficient R idiom

```
> f[f] <- seq_along(f)
```

Such idiom is made possible by the fact that, in R, array indexing starts at one, not zero. If we wish to determine, say, $f(f(f(\cdot))) = f^3(\cdot)$, then manipulating f in word form is computationally inefficient, and it is more convenient to represent f in *cycle form*:

$$(1987)\,(346)$$

which is a compact representation of the fact that $1 \xrightarrow{f} 9 \xrightarrow{f} 8 \xrightarrow{f} 7 \xrightarrow{f} 1$ and $3 \xrightarrow{f} 4 \xrightarrow{f} 6 \xrightarrow{f} 3$, the remaining elements mapping to themselves. Then it is clear that f^3 is the cycle (1789). The R idiom for this would be:

```
> as.cycle(f)
```

```
[1] (1987)(364)
```

```
> f^3
```

[1] By default the package print method coerces words to cycle form but here option `print_word_as_cycle` is changed to show the permutation in word form.

```
[1] (1789)
```

In the package, a cycle is represented internally as a list of integers (the term used in the documentation is "cyclist"), while words are integer matrices; a discussion is given in `cyclist.Rd`. The structure of a typical permutation in cycle form may be seen using the `dput()` function:

```
> dput(f)

structure(c(9L, 2L, 6L, 3L, 5L, 4L, 1L, 7L, 8L), .Dim = c(1L,
9L), class = c("permutation", "word"))

> dput(as.cycle(f))

structure(list(list(c(1L, 9L, 8L, 7L), c(3L, 6L, 4L))), class = c("permutation",
"cycle"))
```

Cycle form for permutation has a number of advantages, not least of which is efficient and readable representation of permutations with only a small number of non-fixed elements.

However, converting between word and cycle form is expensive in the package, and this motivates much of the package design philosophy: objects are coerced from one form to the other only when necessary. Some operations, such as permutation products, are easily carried out in word form; some, such as integer powers of permutations, are more natural in cycle form. Note that inversion is reasonably direct, the idiom for inverting a cyclist `cyc` being

```
> lapply(cyc,function(o){c(o[1],rev(o[-1]))})
```

Permutations in cycle form are difficult to handle in R for two reasons. Firstly, cycles must be well-formed, and this places strict specifications on the R objects: individual bracketed cycles can contain no repeated elements, and must be pairwise disjoint. These conditions must be verified before an object of class `cycle` may be created. Secondly, there are many equivalent ways to represent the same permutation and the package includes substantial amount of code to coerce cycle-form permutations into a canonical representation; an extended discusssion is given in `cyclist.Rd`.

2.1. Multiplication of permutations

Given f and another permutation g, we may combine f and g in two ways: we may perform f first and then g, or alternatively g first and then f; in general, these two products are different.

```
> g <- as.word(9:1)
> f*g

    {1} {2} {3} {4} {5} {6} {7} {8} {9}
[1] .   8   4   7   .   .   9   3   2

> g*f
```

```
    {1} {2} {3} {4} {5} {6} {7} {8} {9}
[1] 8   7   1   .   .   3   6   2   .
```

Word products use the natural R idiom

```
> f*g == g[f]
```

again depending directly on one-based indexing used by R. Note the confusing order of operations: on the left hand side, f appears first and then g; on the right hand side the terms appear in the opposite order, ultimately due to prefix notation combined with the syntatic sugar of the extraction operator `[()`. An extended discussion of this issue is given in `as.function.permutation.Rd` in the package; but the notation used here is partly motivated by the preservation of associativity, in the sense that `a*(b*c) == (a*b)*c` for any three permutations `a,b,c`.

One measure of the non-commutativity of f and g is the *commutator*, here defined as $f^{-1}g^{-1}fg$:

```
> commutator(f,g)
```

```
    {1} {2} {3} {4} {5} {6} {7} {8} {9}
[1] 4   3   6   7   .   9   8   1   2
```

Because working with permutations in cycle form is more natural and compact than word form, the package allows control over the print method via `options()`:

```
> options(print_word_as_cycle = TRUE)
> commutator(f,g)
```

```
[1] (1478)(2369)
```

```
> commutator(g,f)
```

```
[1] (1874)(2963)
```

2.2. The identity element

The permutation that leaves all elements fixed is known as the *identity*. Taking the product of a permutation with its own inverse gives the identity, as does raising any permutation to the power zero:

```
> f * inverse(f)
```

```
[1] ()
```

The print method for cycles shows an empty pair of round brackets symbolising the fact that this permutation does not move any elements.

The identity element is a special case in the package in several respects. Firstly, the identity permutation is problematic when expressed in word form: it has no natural size (the `size()` of a permutation is the cardinality of its ground set) apart from zero. In cycle form, the identity is stored essentially as `list(list())`: the group-theoretic cycles are empty.

2.3. Conjugation and vectorization

The package is vectorized, and this allows the use of a range of natural R idiom. Suppose we wish to consider the symmetry group of a tetrahedron known to be the even permutations of a set of four elements:

```
> S4 <- allperms(4)
> A4 <- S4[is.even(S4)]
> A4

 [1] ()       (234)    (243)    (12)(34) (123)    (124)    (132)    (134)
 [9] (13)(24) (142)    (143)    (14)(23)
```

Note that it is the print method that coerces permutations to cycle form: the package does not do so unless required, being a slow and memory-intensive process. Thus `S4` is all permutations of size 4, and `A4` just the even permutations, known as the alternating group.

As a final illustration, we may calculate the conjugate[2] of the even permutations shown above with a cycle on five elements:

```
> A4^cyc_len(5)

 [1] ()       (345)    (354)    (23)(45) (234)    (235)    (243)    (245)
 [9] (24)(35) (253)    (254)    (25)(34)
```

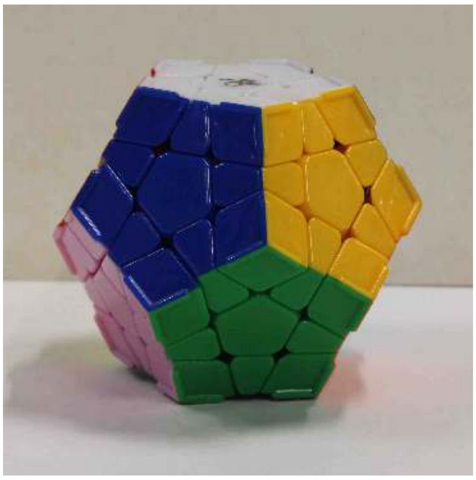
See how the shape of the permutations is unaltered by the conjugation; documentation is a at `shape.Rd` and `conjugate.Rd`.

3. The Megaminx

The *megaminx* is a dodecahedral puzzle with similar construction to the Rubik cube (Joyner 2008); see Figure 1. The puzzle has 50 movable pieces (compare 20 for the Rubik cube) and $12 \times 11 = 132$ coloured stickers ("facets"). When considering the megaminx, it is natural to consider permutations of the facets rather than the pieces, because the pieces may assume multiple orientations.

The permutations package may be used to manipulate the megaminx by assigning each facet a number from 1-129 (see Figure 2) and using the `megaminx` object supplied with the package:

[2] The conjugate of x and y, written x^y, is defined as $y^{-1}xy$; the notation is motivated by the fact that $x^z y^z = (xy)^z$ and $x^{yx} = (x^y)^z$



(a) Megaminx in START position

(b) The superflip

Figure 1: The megaminx

```
> data(megaminx)
> megaminx
                                                                          White
          (10,12,14,16,18)(11,13,15,17,19)(21,33,45,57,69)(22,34,46,58,60)(23,35,47,59,61)
                                                                         Purple
          (15,67,91,81,35)(16,68,92,82,36)(17,69,93,83,37)(20,22,24,26,28)(21,23,25,27,29)
                                                                     DarkYellow
          (17,29,89,79,47)(18,20,80,70,48)(19,21,81,71,49)(30,32,34,36,38)(31,33,35,37,39)
                                                                       DarkBlue
       (10,32,78,118,50)(11,33,79,119,51)(19,31,77,117,59)(40,42,44,46,48)(41,43,45,47,49)
                                                                            Red
    (11,43,115,105,61)(12,44,116,106,62)(13,45,117,107,63)(50,52,54,56,58)(51,53,55,57,59)
                                                                      DarkGreen
       (13,55,103,93,23)(14,56,104,94,24)(15,57,105,95,25)(60,62,64,66,68)(61,63,65,67,69)
                                                                     LightGreen
    (30,88,120,110,40)(31,89,121,111,41)(39,87,129,119,49)(70,72,74,76,78)(71,73,75,77,79)
                                                                         Orange
       (27,99,121,71,37)(28,90,122,72,38)(29,91,123,73,39)(80,82,84,86,88)(81,83,85,87,89)
                                                                      LightBlue
    (25,65,101,123,83)(26,66,102,124,84)(27,67,103,125,85)(90,92,94,96,98)(91,93,95,97,99)
                                                                    LightYellow
(53,113,125,95,63)(54,114,126,96,64)(55,115,127,97,65)(100,102,104,106,108)(101,103,105,107,109)
                                                                           Pink
(41,75,127,107,51)(42,76,128,108,52)(43,77,129,109,53)(110,112,114,116,118)(111,113,115,117,119)
                                                                           Grey
(73,85,97,109,111)(74,86,98,100,112)(75,87,99,101,113)(120,122,124,126,128)(121,123,125,127,129)
```

Object `megaminx` is a 12-element (named) vector of permutations, with elements corresponding to one "click", that is, a 72° clockwise rotation of each face. In practice, it is easier to use abbreviated names for the face turns ("W" for white, "Pu" for purple, and so on).

The package may be used to investigate the effect of consecutive turns on the coloured faces.

For example,

```
> a <- Pu / W * DY^-2 / Pu / DY
```

creates permutation `a` which is the result of turning the purple face one click, then the white face one click counterclockwise, then the dark yellow face two clicks anticlockwise, and so on. The effect of permutation `a` is shown in cycle form by the print method:

```
> a


(10,80,48,20,60,58,46,38,32,36,14,12)(11,81,19,89,35,13)(17,57,45,37,33,71)(18,34)(21,61,5
```

Thus facet 10 moves to position 80, facet 80 moves to position 48, and so on. The order of permutation x is defined as the lowest nonzero integer n for which x^n is the identity:

```
> permorder(a)

Purple
    12
```

showing that `a^12` returns the megaminx to START. This suggests that repeating `a` by a divisor of 12 would produce a pleasing pattern and indeed executing this sequence 6 times gives

```
> a^6

                                      Purple
(10,46)(12,58)(14,60)(20,36)(32,48)(38,80)
```

showing that this has a particularly parsimonious effect.

3.1. The Superflip

One particularly pleasing pattern on the megaminx is the *superflip*, shown in Figure 1b. In this pattern, each edge piece is in its correct position but flipped over. There is an equivalent on the Rubik cube which is of considerable theoretical interest **?**: the centre of the cube group is known to contain only the superflip and the identity Joyner (2008).

One challenging task is to accomplish the superflip in the minimum number of turns. Clark (2012) argues that a lower bound for the turning number is 24 and offers an 83-turn sequence:

```
> X <- W / Pu * W * Pu^2 / DY^2
> Y <- LG^(-1) / DB * LB * DG
> Z <- Gy^(-2) * LB / LG /Pi / LY
> superflip83 <- (X^6)^Y + Z^9  # superflip (Jeremy Clark)
```

(note the use of binary "+" showing that sequences `(X^6)^Y` and `Z^9` commute). It is intuitively clear that the superflip commutes past any sequence of operations, and is thus in the *center* of the megaminx group. Computational group theoretic software such as GAP can be used to show that the megaminx center comprises only the superflip and the identity.

That the superflip is in the center may be verified directly in R using the **permutations** package:

```
> jj <- permprod(sample(megaminx,30,replace=TRUE))
> stopifnot(jj*superflip83 == superflip83*jj)
```

The permutations package may be used to search systematically for a superflip with fewer moves than Clark's 83 turns, and a slight improvement is possible. The following sequence, which required an extensive computer search, exhibits an 82-turn superflip:

```
> superflip82 <-
+     LB^-1 * Gy^-1 * LB^-1 * O^3 * Gy * LY^2 * Gy^2 *
+         LY^3 * Gy^3 * LY^3/LB * Gy^2 *
+             ((Pu^-1 * W^2 * DY * DB * R)^9)^(O^3 * LB^3/LG) *
+                 Gy^2/LB * O^3 * Gy * LY^2 * Gy^2 * LY^3 * Gy^3 *
+                     LY^3 / LB / Gy / LB *
+                     O^3 * Gy * LY^2 * Gy^2 * LY^3 * Gy^3 * LY^3
> stopifnot(superflip82 == superflip83)
```

References

Averbach B, Chein O (2000). *Problem solving through recreational mathematics*. Dover.

Clark J (2012). "TwistyPuzzles.com topic - Megaminx Superflip." URL http://twistypuzzles.com/~sandy/forum/viewtopic.php?f=8&t=24479.

GAP (2016). *GAP – Groups, Algorithms, and Programming, Version 4.8.6*. The GAP Group. URL http://www.gap-system.org.

Joyner D (2008). *Adventures in Group Theory*. John Hopkins Press.

Milne JS (2013). "Group Theory (v3.13)." Available at www.jmilne.org/math/.

Sage (2017). *SageMath, the Sage Mathematics Software System (Version 7.5.1)*. The Sage Developers. URL http://www.sagemath.org.

Stanley RP (2011). *Enumerative Combinatorics, volume I*. Cambridge University Press.

Affiliation:

Robin K. S. Hankin
AUT University

Auckland
New Zealand
E-mail: hankin.robin@gmail.com

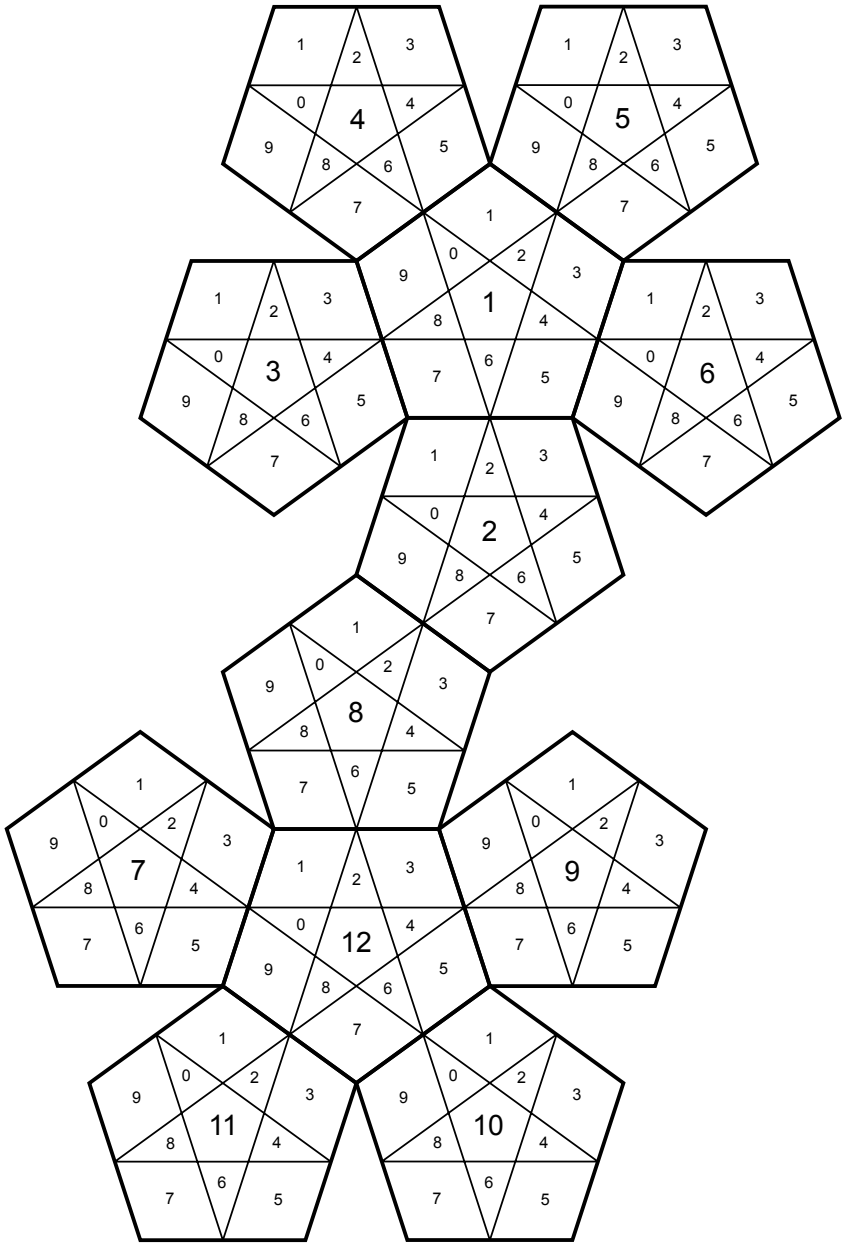


Figure 2: Megaminx net. showing the facet numbering scheme. In each pentagonal face, the facet number is given by ten times the central large number, and the unit is given by the small figure. Thus the top left facet is number 41 and the bottom right facet is 105