Gray Codes for Involutions

Timothy Walsh
Department of Computer Science, University of Quebec in Montreal (UQAM)
P.O. Box 8888, Station A, Montreal, Quebec, Canada H3C-3P8
walsh.timothy@uqam.ca

AMS Classification: 68R05, 05A05
Key words: *involutions, Gray codes, non-recursive sequencing, loop-free sequencing*

Abstract: A Gray code is a list of words such that each word differs from its successor by a number of letters which is bounded independently of the length of the word. We use Roelants van Baronaigien's I-code for involutions to derive a Gray code for all length-n involutions and one for those with a given number of length-2 cycles. In both Gray codes, each involution is transformed into its successor via one or two transpositions or a rotation of three elements. For both Gray codes we obtain algorithms for passing between a word and its position in the list and a non-recursive sequencing algorithm (transforming a given word into its successor or determining that it is the last word in the list) which runs in $O(n)$ worst-case time and uses $O(1)$ auxiliary variables; for involutions with a given number of length-2 cycles we also obtain a sequencing algorithm which runs in $O(1)$ worst-case time and uses $O(n)$ auxiliary variables. We generalize Chase's method for obtaining non-recursive sequencing algorithms to any list of words in which all the words with a given suffix form an interval of consecutive words, and we show that if in addition the letter preceding the suffix always takes at least two distinct values in that interval, then Ehrlich's method will find in $O(1)$ time the rightmost letter in which a word differs from its successor.

0. Introduction

A *length-n involution* is a permutation $(p_1,p_2,...,p_n)$ of $\{1,...,n\}$ such that for each i, $1\le i\le n$, either $p_i=i$ or else $p_i=j\ne i$ and $p_j=i$. An element p_i is a *fixed point* of the involution if $p_i=i$.

Example 1: The permutation (4,2,3,1,7,6,5,8)=(1,4)(2)(3)(5,7)(6)(8) is an involution with four fixed points: 2, 3, 6 and 8.■

A *fixed-point-free* involution, or FPFI, is an involution with no fixed points. Algorithms for generating all the involutions of length n in lexicographical order were presented in [7] and [11]; [7] also contains algorithms for generating all the fixed-point-free involutions of length $2n$ in lexicographic order. Roelants van Baronaigien's I-code for length-n involutions [11] turns out to preserve not only lexicographical order but also closeness, so that to obtain a Gray code on the set of all involutions, or on the set of FPFIs, it suffices to obtain a Gray code on the set of I-codes for the corresponding set of involutions.

In Section 1 we present a Gray code for FPFIs such that each FPFI is transformed into the next one via two transpositions - two cycles (x,y) and (i,j) with $x<y$ and $|y-j|=1$ are replaced by the cycles (x,j) and (i,y), so that in one-line notation $p_y=x$ swaps with its immediate neighbour $p_j=i$ and $p_x=y$ swaps with $p_i=j$. This Gray code has an application to combinatorial map theory because FPFIs represent rooted one-vertex maps on orientable surfaces [3].

In section 2 we present a Gray code for length-n involutions with k length-2 cycles (and $n-2k$ fixed points) such that each involution is transformed into its successor by either two transpositions or a rotation of three elements. This Gray code is obtained by combining the Gray code for FPFIs presented in section 1 with the Eades-McKay Gray code [5] for k-combinations of $\{1,...,n\}$.

In Section 3 we present a Gray code for all length-n involutions such that each involution is transformed into its successor by either one transposition, two transpositions or a rotation of three elements.

The *rank* of an item w of a list L is the number of items preceding w in L, and to *sequence* w is to find the successor to w in L or to determine that w is the last item of L. For all these Gray codes we give ranking algorithms as well as non-recursive sequencing algorithms which run in $O(n)$ worst-case time and use $O(1)$ auxiliary variables. We also give sequencing algorithms which run in $O(1)$ worst-case time and use $O(n)$ auxiliary variables for the Eades-McKay Gray code for combinations (described recursively in [5]) and our own Gray codes for FPFIs and for involutions with k length-2 cycles. All the sequencing algorithms have been programmed in C and tested (listings are available on request); we report the time-trials we conducted.

The non-recursive sequencing algorithms were derived using a method described by Chase [2] for word lists in a generalized lexicographical order which he called Graylex order; here we generalize it further to any list of words in which all the words with a given suffix form an interval of consecutive words. Almost all the Gray codes in the literature have this property or can be transformed into one that does by left-right reversal and/or position vectors, a notable exception being Savage's Gray code for integer partitions [14]. We also show that the auxiliary array used by Erhlich [6] to determine in constant time the rightmost letter which must change to sequence a given word will work for any list of words with the above property provided that in addition the letter preceding the suffix always takes at least two distinct values in that interval. The I-code for FPFIs in Gray code order has this additional property; the Eades-McKay Gray code does not, but sequencing can be made loop-free using an array-implemented stack. The I-

code for all length-n involutions in Gray code order does not have this property, and although it too can be sequenced in $O(1)$ time using an auxiliary array, loop-free sequencing of the induced Gray code for all length-n involutions is still an open problem.

1. Fixed-point-free involutions

The I-code for involutions [11] restricted to FPFIs is a lexicographical-order-preserving bijection from the set of $(2n\text{-}1)\times(2n\text{-}3)\times...\times3\times1$ length-$2n$ FPFIs in one-line notation onto the Cartesian product of integer intervals $[1,2n\text{-}1]\times[1,2n\text{-}3]\times...\times[1,3]\times[1,1]$. By subtracting 1 from each n-tuple and then dropping its last member, which is 0, we obtain the following lexicographical-order-preserving bijection R from the set of length-$2n$ FPFIs onto $[0,2n\text{-}2]\times[0,2n\text{-}4]\times...\times[0,2]$. Let $(x(1),y(1))...(x(n),y(n))$, where $x(i)<y(i)$ for all i and $x(1)<...<x(n)$, be the *canonical* cycle notation for an FPFI. For each i, $x(i)$ is the smallest element in $\{1,...,2n\}\text{-}\{x(1),y(1),...,x(i\text{-}1),y(i\text{-}1)\}$ and there are $2n\text{-}2i+1$ elements in the set

$$S(i) = \{1,...,2n\} - \{x(1),y(1),...,x(i\text{-}1),y(i\text{-}1),x(i)\} \tag{1}$$

from which to choose $y(i)$. The function R maps the FPFI $(x(1),y(1))...(x(n),y(n))$ onto $(g_1,...,g_{n\text{-}1})$, where g_i is the number of elements of $S(i)$ which are smaller than $y(i)$ (see the first two columns of Table 1).

The lexicographical-order rank of the word $(g_1,...,g_{n\text{-}1})$ is just the number $g_1...g_{n\text{-}1}$, where each g_i is in radix $2n\text{-}2i+1$. Passing between the (modified) I-code and its rank can be done in $O(n)$ arithmetic operations including multiplying a small integer by a large one. Passing between an FPFI F and $R(F)$ takes $O(n^2)$ elementary operations if it takes $O(n)$ time to remove an element from the indexed set $S(i)$. This estimate can be reduced to $O(n \log n)$ if the set $S(i)$ is updated using a balanced tree or the more space-efficient algorithms in [9, pp. 578-579 (answers to exercises 5 and 6, p. 19] and $O(\text{n log } n/\log\log n)$ using a more sophisticated data structure [4].

Example 2: Let $F = (f_1,...,f_n) = (7,4,6,2,8,3,1,5)$. Then $x(1)=1$ and $S(1)=(2,3,4,5,6,7,8)$. Since $y(1)=f_1=7$, which is greater than 5 of the elements of $S(1)$, $g_1=5$. Then $x(2)=2$ and $S(2)=(3,4,5,6,8)$. Since $y(2)=f_2=4$, $g_2=1$. Then $x(3)=3$ and $S(3)=(5,6,8)$. Since $y(3)=f_3=6$, $g_3=1$. Thus $R(F)=(5,1,1)$, so that the lexicographical-order rank of F is 511 in base (7,5,3), which is $((5\times3)+1)\times5+1=81$. Conversely, knowing the rank 81 of F we can extract $R(F)=(5,1,1)$, and by successive calculation of $S(i)$ we can compute F.■

The function R preserves closeness as well as lexicographic order: if g_i is increased by 1, then $y(i)$ must rise to the next larger value j in the set $S(i)$. In cycle notation, the two cycles $(x(i),y(i))$ and (j,p_j) are replaced by $(x(i),j)$ and $(y(i),p_j)$, so that in one-line notation $y(i)$ swaps values with the element j, while $x(i)$ swaps positions with p_j, the nearest element to the right of $x(i)$ which is in the set $S(i)$, or, equivalently, which is larger than $x(i)$.

If we use the Gray code in [19, p. 112] for the set of $(g_1,...,g_{n-1})$, where g_1 varies the slowest, then $x(i)$ will not always swap with its immediate neighbour - for example, when $R(F)$ passes from 20 to 21, F will pass from 432165 to 456123 - so that finding in $O(1)$ time the element with which $x(i)$ is to swap is a non-trivial problem. If we instead allow g_1 to vary the fastest, then, when g_i changes, all the g_j, $j<i$, are at their extreme values, so that $S(i)$ is an interval. Thus $y(i)$ must change by 1 and $x(i)$ must swap with its immediate neighbour. Thus we choose the latter Gray code for the set of words $(g_1,...,g_{n-1})$, $0\le g_i\le 2(n-i)$, and we define $\mathsf{f}(2n)$ to be the corresponding Gray code induced on the set of FPFIs of $\{1,...,2n\}$ (see the last two columns of Table 1). The first value of F is $(2,1,4,3,...,2n,2n-1)$ and $R(F)=(0,...,0)$.

PUT TABLE 1 ABOUT HERE

All the words with a given suffix $(g_{i+1},...,g_{n-1})$ form an interval of consecutive words in which g_i takes the sequence of distinct values $(0,1,...,2(n-i))$ if $g_{i+1}+...+g_{n-1}$ is even and $(2(n-i),...,1,0)$ otherwise (in the former case we say that g_i is *increasing* and in the latter, that it is *decreasing*). The rank of F in $\mathsf{f}(2n)$ is obtained by replacing each decreasing g_i in $R(F)$ by $2(n-i)-g_i$ and then evaluating the resulting word as a mixed-radix integer in co-lexicographical order (right-to-left), so that (continuing example 2) the word (5,1,1) corresponds to the rank $((1\times5)+(4-1))\times7)+5=61$.

The first word $(g_1,...,g_{n-1})=(0,...,0)$. Each word $(g_1,...,g_{n-1})$ can be transformed into its successor using the generic sequencing algorithm listed as Algorithm 1, which generalizes the sequencing algorithm used in [2], [19] and many other places. It works for any list of words $(g_1,...,g_{n-1})$ in which all the words with a common suffix $(g_{i+1},...,g_{n-1})$ form an interval of consecutive words in the list, so that this interval is partitioned into sub-intervals by the sequence of distinct values assumed by g_i (a proof of this observation can be found in [15]). It was shown in [2] to work whenever all the sequences are monotone (Graylex order) but has been implicitly used for many lists in which this condition does not hold, such as the Gray code for set partitions in [6].

PUT ALGORITHM 1 ABOUT HERE

For each word w in a list except the last one, the *pivot* is the index of the rightmost letter (the *pivotal element*) that must change to transform w into its successor. In the above Gray code only one g_i actually changes - all the g_j, $j<i$, are now at their first values with respect to the new

suffix - and this change in $R(F)=(g_1,...,g_{n-1})$ induces a corresponding pair of transpositions in the FPFI F.

Example 3: If $F=(8,4,6,2,7,3,5,1)$, then $R(F)=(g_1,g_2,g_3)=(6,1,1)$. Since g_2+g_3 is even, g_1 takes the sequence of values (0,1,2,3,4,5,6); so g_1 is already at its final value. Since g_3 is odd, g_2 takes the sequence of values (4,3,2,1,0); so 2 is the pivot and g_2 drops to 0, changing $R(F)$ to (6,0,1) and F to (8,3,2,6,7,4,5,1).■

Example 4: If $F=(2,1,4,3,7,8,5,6)$, then $R(F)=(g_1,g_2,g_3)=(0,0,1)$. Since g_2+g_3 is odd, g_1 takes the sequence of values (6,5,4,3,2,1,0); so g_1 is already at its final value. Since g_3 is odd, g_2 takes the sequence of values (4,3,2,1,0); so g_2 too is at its final value. The suffix for g_3 is empty, with sum 0; so g_3 takes the sequence of values (0,1,2). Thus 3 is the pivot and g_3 rises to 2, changing $R(F)$ to (0,0,2) and F to (2,1,4,3,8,7,6,5).■

Once the generic sequencing algorithm is specialized to a particular list it can be optimized in various ways. To avoid calculating the parity of $g_{i+1}+...+g_{n-1}$ for each new i we observe that for each $j<i$, g_j is either 0 or $2(n-j)$, which are both even, so that g_i is increasing if and only if it has the same parity as $g_1+...+g_{n-1}$. We store a Boolean variable *Odd* which is true if $g_1+...+g_{n-1}$ is odd; *Odd* is initialized to False for the first word (0,...,0) and changes value for each successive word. Now the first, next and last value for each g_i can be computed in $O(1)$ time, so that the sequencing algorithm for a length-n FPFI F runs in $O(n)$ time and uses a size-n auxiliary array for $R(F)$.

To obtain a sequencing algorithm for $\mathsf{f}(2n)$ without having to store $R(F)$ as an auxiliary array we use the following theorem.

Theorem 1: Let $F=(f_1,...,f_n)$ and let i be the pivot of $R(F)$. If *Odd* is True, then $x(i)=2i-1$, $g_i=y(i)-2i$, $F=(2,1,4,3,...,2i-2,2i-3,y(i)>2i,...)$ and i is the smallest value such that $f_{2i-1}>2i$; otherwise $x(i)=i$, $g_i=y(i)-(i+1)$, $F=(2n,2n-1,...,2n-i+2,y(i)<2n-i+1,...,i-1,...,2,1)$ and i is the smallest value such that $f_i<2n-(i-1)$ if there is one.

Proof: We recall that each g_j, $j<i$, is at its extreme value, which is even. If *Odd* is True, then g_i is the leftmost letter of $R(F)$ which is not 0. By the definition of the function R, $x(1)=1$, $y(1)=2$, $x(2)=3$, $y(2)=4$, ... , $x(i-1)=2i-3$, $y(i-1)=2i-2$, $x(i)=2i-1$ but $y(i)>2i$, leading to the first expression for F and thence to the first expression for i. The set $S(i)-\{x(i)\}$ is the interval $[2i,2n]$, so that $y(i)=g_i+2i$. If *Odd* is False, then g_i is the leftmost letter of $R(F)$ which is not $2(n-i)$, if there is one. By the definition of R, $x(1)=1$, $y(1)=2n$, $x(2)=2$, $y(2)=2n-1$,...,$x(i-1)=i-1$, $y(i-1)=2n-$

i+2, $x(i)=i$ but $y(i)<2n-i+1$, leading to the second expression for F and thence to the second expression for i. The set $S(i)-\{x(i)\}$ is the interval $[i+1,2n-i+1]$, so that $y(i)=g_i+i+1$.■

We illustrate the sequencing algorithm by continuing examples 3 and 4.

In example 3, where $F=(8,4,6,2,7,3,5,1)$, $R(F)=(6,1,1)$ (which isn't stored); so *Odd* is False. Now $f_1=8$ but $f_2<7$; so $i=2$. Then $x(2)=2$, $y(2)=f_2=4$ and $g_2=4-(2+1)=1$ which is odd; so it is decreasing. Thus $f_4=2$ trades places with its immediate left neighbour $f_3=6$ and their mates $f_2=4$ and $f_6=3$ also trade places, so that F changes to $(8,3,2,6,7,4,5,1)$.■

In example 4, where $F=(2,1,4,3,7,8,5,6)$, $R(F)=(0,0,1)$ (which isn't stored); so *Odd* is True. Now $f_1=2$ and $f_3=4$ but $f_5>6$; so $i=3$. Then $x(3)=5$, $y(3)=f_5=7$ and $g_3=7-2\times3=1$ which is odd; so it is increasing. Thus $f_7=5$ trades places with its immediate right neighbour $f_8=6$ and their mates $f_5=7$ and $f_6=8$ also trade places, so that F changes to $(2,1,4,3,8,7,6,5)$.■

The algorithm maintains only F, *Odd* and another Boolean variable *Done* which is initialized to False and becomes True when the FPFI turns out to be the last one; so it uses $O(1)$ auxilliary variables but it takes $O(n)$ time in the worst case to find the pivot i by scanning F from left to right. This search can be avoided by using the auxiliary array $e=(e_1,...,e_n)$ introduced in [1] and [6] and generalized in [19, p. 112] and elsewhere.

We generalize it further to any word list in which all the words with a common suffix form an interval of consecutive words in which the letter preceding the suffix takes at least two distinct values. Denote by a_i, z_i and h_i the first value, last value and successor to g_i, respectively, in the sequence of distinct values assumed by g_i as a function of the suffix $(g_{i+1},...,g_{n-1})$. **Since g_i takes at least two distinct values, a_i is never equal to z_i.** A *z-word* is a subword $(g_j,...,g_k)=(z_j,...,z_k)$ which is maximal by inclusion: either $j=1$ or $g_{j-1}\neq z_{j-1}$, and either $k=n-1$ or $g_{k+1}\neq z_{k+1}$. From Algorithm 1 it is clear that the pivot is the smallest value of i such that $g_i\neq z_i$. If $g_1\neq z_1$, then 1 is the pivot. If $(g_1,...,g_{i-1})$ is a z-word, where $i<n$, then i is the pivot. If $(g_1,...,g_{n-1})=(z_1,...,z_{n-1})$, then it is the last word in the list and we call n the pivot. The generic sequencing algorithm with loop-free pivot-finding is listed as Algorithm 2. We will show that **for each j, $e_j=j$ unless $(g_j,...,g_{k-1})$ is a z-word for some $k>j$, in which case $e_j=k$**. From this it follows that e_1 is always the pivot. The first word is $(a_1,...,a_{n-1})$, and since $a_i\neq z_i$ for all i there are no z-words. Since $(e_1,...,e_n)$ is initialized to $(1,...,n)$ the formula for e holds initially, and the comments in Algorithm 2 imply that if it holds before the algorithm is executed then it will hold afterwards, so that it always holds.

PUT ALGORITHM 2 HERE OR HIGHER

The list of I-codes for FPFIs satisfies the condition under which Algorithm 2 works. Using Theorem 1 we implemented a loop-free sequencing algorithm for FPFIs without storing the auxiliary array $R(F)$ (see Algorithm 3). The time trials indicated that neither the elimination of the auxiliary array $R(F)$ nor the use of the auxiliary array e to make sequencing loop-free had any significant effect on execution time.

PUT ALGORITHM 3 ABOUT HERE

2. Involutions with a given number of length-2 cycles

We recall that $\mathsf{f}(2n)$ is the list of all the length-$2n$ FPFIs in the order given by the Gray code of Section 1 and we denote by $\mathsf{f}^R(2n)$ the same list in reverse order.

Let $C=(c_1,c_2,...,c_{2k})$, where $c_1<...<c_{2k}$, be a $2k$-combination (subset) of $\{1,2,...,n\}$ and F be an FPFI $(f_1,f_2,...,f_{2k})$. Denote by $P(C,F)$ the involution whose 1-cycles are the elements of $\{1,...,n\}$-C and whose 2-cycles are (c_i,c_j) for each 2-cycle (i,j) of F. Any involution can be uniquely expressed as $P(C,F)$ for some combination C and some FPFI F.

Example 5: If C is the combination (1,4,5,7) of $\{1,...,8\}$ and F is the FPFI (2,1,4,3), then $P(C,F)$ is the involution (4,2,3,1,7,6,5,8).■

Given any $2k$-combination C, we denote by $P(C,\mathsf{f}(2k))$ the list of involutions $P(C,F)$ as F runs over $\mathsf{f}(2k)$, with a similar definition for $P(C,\mathsf{f}^R(2k))$. Given a list $\mathsf{c}(n,2k)=(C_0,C_1,...)$ of $2k$-combinations of $\{1,...,n\}$, we denote by $\mathsf{p}(\mathsf{c}(n,2k),\mathsf{f}(2k))$ the list

$$P(C_0,\mathsf{f}(2k))\mathrm{o}P(C_1,\mathsf{f}^R(2k))\mathrm{o}P(C_2,\mathsf{f}(2k))\mathrm{o}...,$$

where o is the concatenation operator for lists (we run through the Gray code for FPFIs alternately forwards and backwards for each successive combination).

Two adjacent involutions in the same sublist $P(C_{2a},\mathsf{f}(2k))$ or $P(C_{2a+1},\mathsf{f}^R(2k))$ will differ by two transpositions: if in F the cycles (x,y) and (i,j) with $x<y$ and $|y-j|=1$ are replaced by (x,j) and (i,y), then in $P(C_{2a},F)$ or $P(C_{2a+1},F)$ the cycles (c_x,c_y) and (c_i,c_j) will be replaced by (c_x,c_j) and (c_i,c_y), so that in one-line notation c_x swaps with c_i and c_y swaps with c_j, and each element m between c_x and c_i will satisfy $p_m=m$.

If P and its successor P' in $\mathsf{p}(\mathsf{c}(n,2k),\mathsf{f}(2k))$ are in adjacent sublists, then P must be $P(C_a,F)$ and P' must be $P(C_{a+1},F)$, where F is either the last or the first FPFI in $\mathsf{f}(2k)$ depending upon whether a is even or odd. The difference between P and P' will, of course, depend upon the difference between adjacent words in c, but a minimal difference between C_a and C_{a+1} **as subsets** does not guarantee a minimal difference between P and P'. As an extreme example, suppose that $n=2k+1$, $C_a=(1,2,3,...,2k)$, $C_{a+1}=(2,3,...,2k,2k+1)$ and $F=(2,1,4,3,...,2k,2k-1)$. Now C_{a+1} differs from C_a by the minimal amount - one element swapped out and one element swapped in - but $P=(2,1,4,3,...,2k,2k-1,2k+1)$ and $P'=(1,3,2,5,4,...,2k+1,2k)$ differ in all $2k+1$ elements because the element 1 swapped out of C_a and the element $2k+1$ swapped in have $2k-1$

elements in C_a between them (larger than 1 but smaller than $2k+1$). Even a single element in C_k between the element swapped in and the element swapped out, as occurs in both the Liu-Tang Gray code [10] for combinations and the Gray code in [2], results in a rotation of 5 elements in $P(C_a,F)$ (see the example below).

With the Eades-McKay Gray code [5], on the other hand, C_a contains no elements between the element *old* swapped out and the element *new* swapped in (and this is optimal in the sense that for some values of n and k there is no Gray code in which the element swapped in and the element swapped out differ by 1 [12]). Then *old* will simply be replaced by *new* in the sorted list C_a. For any FPFI F, $P(C_a,F)$ will be transformed into $P(C_{a+1},F)$ by replacing the two cycles (*old*,*mate*), (*new*) by the two cycles (*old*), (*new*,*mate*), so that p_{old} changes from *mate* to *old*, p_{mate} from *old* to *new* and p_{new} from *new* to *mate* and in one-line notation the transformation made is the rotation $(mate,old,new) = (p_{mate},p_{new},p_{old})$. The corresponding Gray code for involutions $\mathsf{p}(n,k) = P(\mathsf{c}(n,2k),\mathsf{f}(2k))$ will have thus the property that any involution is transformed into its successor by either two transpositions or a rotation of three elements.

The Eades-McKay Gray code $\mathsf{c}(n,k)$ for k-combinations of $\{1,...,n\}$ (modified by reversing each word, reversing the whole list and subtracting each number from $n+1$) is described recursively by

$$\mathsf{c}(n,k) = \mathsf{c}(n\text{-}1,k) \text{ o } \mathsf{c}^R(n\text{-}2,k\text{-}1)n \text{ o } \mathsf{c}(n\text{-}2,k\text{-}2)(n\text{-}1)n, \qquad (2)$$

anchored by $\mathsf{c}(k,k)=(1,2,3,...,k)$, $\mathsf{c}(n,1)=(1)o(2)o...o(n)$ and $\mathsf{c}(n,0)$ = the empty list. See Table 2 for a comparison of the Eades McKay Gray code with the Liu-Tang Gray code.

PUT TABLE 2 ABOUT HERE

Continuing example 5, the involution $(1,4)(2)(3)(5,7)(6)(8)=(4,2,3,1,7,6,5,8)=P(C,F)$, where C is the combination (1,4,5,7) of $\{1,...,8\}$ and F is the FPFI $(1,2)(3,4)=(2,1,4,3)$, the first length-4 FPFI in $\mathsf{f}(4)$. The next combination after C in the Liu-Tang Gray code is C'=(1,2,4,7), and $P(C',F)$=(1,2)(3)(4,7)(5)(6)(8)=(2,1,3,7,5,6,4,8), which is obtained from (4,2,3,1,7,6,5,8) by the 5-element rotation (4,2,1,7,5). On the other hand, the next combination after C in the Eades-McKay Gray code is C''=(1,2,5,7), and $P(C'',F)$=(1,2)(3)(4)(5,7)(6)(8)=(2,1,3,4,7,6,5,8), which is obtained from (4,2,3,1,7,6,5,8) by the 3-element rotation (4,2,1).■

To obtain a non-recursive description of $\mathsf{c}(n,k)$ we define $run(i)$, for each i, to be the maximum value of j such that $c_{i+1},...,c_{i+j}$ are consecutive integers, and we use the following theorem. Note that whenever $run(i)$ is even the Eades-McKay Gray code follows the same rule shown in [2] to be followed by the Liu-Tang Gray code.

Theorem 2: All the combinations with the same suffix $(c_{i+1},...,c_k)$ form an interval of consecutive words in which c_i takes the sequence of distinct values

$i,i+1,...,m-1,m$ if $k-i$ is even and $run(i)$ is even,

$m,m-1,...,i+1,i$ if $k-i$ is odd and $run(i)$ is even,

$m,i,i+1,...,m-1$ if $k-i$ is even and $run(i)$ is odd,

$m-1,...,i+1,i,m$ if $k-i$ is odd and $run(i)$ is odd,

where m is equal to n if $i=k$ and $c_{i+1}-1$ otherwise.

Proof. This proposition is true for the anchoring lists $\mathtt{C}(k,k)=(1,2,3,...,k)$, $\mathtt{C}(n,1)=(1)o(2)o...o(n)$ and $\mathtt{C}(n,0)$=the empty list. Suppose that $k\geq 2$ and $n\geq k+1$ and that the proposition is true for the lists $\mathtt{C}(n-1,k)$, $\mathtt{C}(n-2,k-1)$ and $\mathtt{C}(n-2,k-2)$.

The sublists $\mathtt{C}(n-1,k)$, $\mathtt{C}^R(n-2,k-1)n$ and $\mathtt{C}(n-2,k-2)(n-1)n$ of $\mathtt{C}(n,k)$ all have the property that all the words with a common suffix form an interval of consecutive words, and this property is passed on to $\mathtt{C}(n,k)$ because $\mathtt{C}(n-1,k)$ consists of all the combinations such that $c_k<n$, $\mathtt{C}^R(n-2,k-1)n$ of those such that $c_k=n$ and $c_{k-1}<n-1$, and $\mathtt{C}(n-2,k-2)(n-1)n$ of those such that $c_k=n$ and $c_{k-1}=n-1$ so that any two words in different sublists can have only the empty suffix in common.

It remains to verify the proposition that the sequence of (distinct) values attained by each c_i in each of those three sublists is the one stated in the theorem.

The last letter c_k takes the sequence of values $k,...,n-1$ in the sublist $\mathtt{C}(n-1,k)$ and the value n in each of the other two sublists; so it behaves as it should, since $run(k)$ is always 0. The second-last letter c_{k-1} behaves as it should in the sublist $\mathtt{C}(n-1,k)$ by the induction hypothesis, and since its suffix is different in this sublist from in the other two we can treat this sublist by itself. In $\mathtt{C}^R(n-2,k-1)n$ it takes the sequence of values $n-2,...,k,k-1$ by the induction hypothesis, and in $\mathtt{C}(n-2,k-2)(n-1)n$ it takes the value $n-1$; so that in $\mathtt{C}^R(n-2,k-1)n$ o $\mathtt{C}(n-2,k-2)(n-1)n$ it takes the sequence of values $n-2,...,k,k-1,n-1$, which satisfies the proposition since $run(k-1)$ is always 1.

For the other letters c_i, $i\leq k-2$, the sublists can be treated separately, since c_i will have a different suffix in each of these sublists.

The proposition is true in the sublist $\mathtt{C}(n-1,k)$ by the induction hypothesis.

The sequence of values attained by c_i for $1\leq i\leq k-2$ in the interval of words with a common suffix is the reverse in $\mathtt{C}^R(n-2,k-1)$ of what it is in $\mathtt{C}(n-2,k-1)$. But when n is appended at the right

of each word in $\mathsf{C}^R(n\text{-}2,k\text{-}1)$ to make $\mathsf{C}^R(n\text{-}2,k\text{-}1)n$, 1 is added to each k, changing the parity of $k\text{-}i$, and $run(i)$ is never changed, which, according to the proposition, reverses the sequence of values attained by c_i. It follows that the proposition holds in $\mathsf{C}^R(n\text{-}2,k\text{-}1)n$.

Appending n-1 and then n to the right of each word in $\mathsf{C}(n\text{-}2,k\text{-}2)$ changes k by 2 and $run(i)$ by either 0 or 2; so the proposition holds in $\mathsf{C}\ (n\text{-}2,k\text{-}2)(n\text{-}1)n$ as well.

The result follows by induction on n.■

We note here that Ruskey [13] proved that the Knuth Gray code [18] for integer compositions is isomorphic to the Eades-McKay Gray code, so that Klingsberg's non-recursive description [8] of Knuth's Gray code could be used to obtain a non-recursive description of the Eades-McKay Gray code.

To optimize the generic sequencing algorithm for this Gray code we use the following theorem.

Theorem 3: Let $r=run(0)$. Then the pivot cannot have any value other than 1, r or r+1. If the pivot is r+1, then c_{r+1} drops to c_r+1.

Proof. If k=1, then the pivot, if it exists, must be 1; so we assume that k>1.

Suppose that r>1, so that $c_1=c_2$-1, and suppose also that 1 is not the pivot, so that c_1 is at its last value. Since c_1 is at its maximum value m as defined in the statement of Theorem 2, either k-1 and $run(1)=r$-1 must have the same parity or else m=1. For each i, $1\le i\le r$-1, $c_i=c_{i+1}$-1, so that c_i is also at its maximum value. If k-1 and $run(1)$ have the same parity, then so do $k\text{-}i$ and $run(i)=r\text{-}i$, and if the maximum value of c_1 is 1 then the maximum value of c_i is i, so that in either case each of these c_i will be at its last value. Therefore the pivot cannot be smaller than r.

Since $r=run(0)$, either $r=k$ or else $c_r<c_{r+1}$-1. If $r=k$, then either $c_r=n$, in which case the combination is the last one, or else r is the pivot. Suppose that $r<k$, so that $c_r<c_{r+1}$-1, and suppose also that r is not the pivot, so that c_r is at its last value. Since c_r is not at its maximum value, it follows that $k\text{-}r$ and $s=run(r)$ must have opposite parity.

Suppose that s is even, so that $k\text{-}r$ is odd, and so $c_r=r$. Then k-(r+1) is even and $run(r+1)=s$-1, which is odd. Also, since $run(r)$>1, c_{r+1} is at its maximum value; so it must jump to r+1, which is c_r+1.

Suppose that s is odd and $s>1$. Then $k-r$ is even, and so $c_r=c_{r+1}-2$. Also $k-(i+1)$ is odd and $run(r+1)=s-1$, which is even, and c_{r+1} is at its maximum value; so it drops by 1 to c_r+1.

Finally, suppose that $s=1$, so that $k-r$ is even. Then $c_{r+1}=c_r+2>r+1$. Also, $k-(r+1)$ is odd; so once again c_{r+1} drops by 1 to c_r+1 unless c_{r+1} is at its maximum value and $run(r+1)$ is odd. But that would imply that $c_{r+1}=c_{r+2}-1$, which contradicts the fact that $run(r)=1$.

In all cases where neither 1 nor r is the pivot and $r<k$, $r+1$ is the pivot and c_{r+1} drops to c_r+1.■

It follows that the generic sequencing algorithm, specialized to the Eades-McKay Gray code for combinations, becomes the algorithm listed as Algorithm 4. The only variable that has to be maintained from one combination to the next is the Boolean variable *Done*, which is true if the current combination is the last one. The first combination is $(1,2,...,k)$ and for this combination *Done* is False.

PUT ALGORITHM 4 ABOUT HERE

Once $r=run(0)$ and $s=run(r)$ have been computed, testing whether c_1 or c_r is at its last value and, if not, changing it to its next value can be done in $O(1)$ time using Theorem 2. It takes $O(k)$ time to compute r and s by scanning the combination, so that the sequencing algorithm runs in $O(k)$ time and uses $O(1)$ auxiliary variables.

The auxiliary array e cannot be used to make the algorithm run in $O(1)$ time because there are proper suffixes which are not common to more than one word (for example, the suffix (4) in Table 2). Instead we store the lengths of the maximal subwords of consecutive integers in the word $(c_1,c_2,...,c_k)$ on a stack, with the lengths r and s of the leftmost two such subwords on the top of the stack. At most the top two elements of the stack and the top-of-stack index have to be updated; so this version of the sequencing algorithm runs in $O(1)$ time. Both versions of this algorithm can easily be modified to generate non-decreasing (rather than strictly increasing) sequences of integers between 1 and n, as well as compositions of n where two adjacent elements change value from one composition to the next (but not always by 1). A loop-free implementation of the Knuth-Klingsberg Gray code for integer compositions (in which two elements which are not always adjacent change by 1) appears in [15].

Time trials indicate that introducing the stack reduces the run-time by about 5% for generating $\mathsf{C}(2k,k)$. We have implemented the Liu-Tang Gray code so that it runs in $O(1)$ worst-case time and uses $O(1)$ auxiliary variables by using the fact that the pivot differs by at most 2

from one combination to the next [15] (see Table 2); this implementation runs more than twice as quickly as either implementation for generating $\mathsf{c}(n,k)$. It too can be modified to sequence integer compositions in $O(1)$ time using $O(1)$ auxiliary variables (but three integers may change). We note here that [2] also sequences combinations in $O(1)$ time and extra space but in a different order.

Combinations can be ranked and unranked in $\mathsf{c}(n,k)$ in $O(n)$ arithmetic operations using the following ranking formula (which is admittedly more complicated than ranking combinations in lexicographical order but is included here for its recreational value):

$$Rank(c_1,c_2,\ldots,c_n) =$$

$$\sum_{i=1}^{k}(-1)^{k-i}\left\{\begin{array}{l}\left\{\begin{array}{c}\binom{c_i}{i} \text{ if } run(i) \text{ is even}\\ \binom{c_{i+1}-2}{i-1}+\left\{\begin{array}{c}\binom{c_i}{i} \text{ if } c_i < c_{i+1}-1\\ 0 \text{ otherwise}\end{array}\right\} \text{otherwise}\end{array}\right\}\\ -(k \bmod 2).\end{array}\right. \qquad (3)$$

The rank of the involution $P(C,F)$ in $P(C_0,\mathsf{f}(2k))\mathrm{o}P(C_1,\mathsf{f}^R(2k))\mathrm{o}P(C_2,\mathsf{f}(2k))\mathrm{o}\ldots$ is $(2k\text{-}1)\times(2k\text{-}3)\times\ldots\times 3$ times the rank r of C in $\mathsf{c}(n,2k) = C_0,C_1,C_2,\ldots$ plus the rank of F in either $\mathsf{f}(2k)$ if r is even or in $\mathsf{f}^R(2k)$ otherwise.

When the sequencing algorithm of Algorithm 3 processes the last word of $\mathsf{f}(2k)$, *Odd* will be False and e will be $(k,2,3,4,\ldots,k)$. To begin generating $\mathsf{f}^R(2k)$ (after changing C to its successor) it suffices to change *Odd* to True and e_1 from k to 1 (*Odd* now refers to the parity of the rank of the involution $P(C,F)$). Thus, we can sequence each involution $P(C,F)$ of $P(C_0,\mathsf{f}(2k))\mathrm{o}P(C_1,\mathsf{f}^R(2k))\mathrm{o}P(C_2,\mathsf{f}(2k))\mathrm{o}\ldots$ in $O(1)$ time if we keep four auxiliary arrays: F, e, C and the stack of run-lengths of C. The array e is of size k and the other three auxiliary arrays are all of size $2k$; so the extra space needed is in $O(k)$, and thus in $O(n)$.

If we are prepared to scan each involution $P=P(C,F)$ we can get enough information about C and F to update P without storing any auxiliary arrays, so that the algorithm runs in $O(n)$ worst-case time but uses only $O(1)$ auxiliary variables. To this end we observe that an element p_i of P is in C iff $p_i \neq i$ and that the form of F as given in Theorem 1 has to be modified to the corresponding FPFI on the elements of C (see [16] for details). This sequencing algorithm ran only 3% slower than the loop-free algorithm that uses four auxiliary arrays.

3. All length-n involutions

The I-code is a lexicographical-order-preserving injection that maps each length-n involution with c cycles onto a c-tuple of non-negative integers, except that if $c=n$ then the last member of the n-tuple, which is a zero, is dropped. A general involution P will have 1-cycles $(x(i))$ as well as 2-cycles $(x(i),y(i))$. To make the cycle notation canonical, we insist that $x(i)<y(i)$ and that $x(1)<...<x(c)$. If $(x(i))$ is a 1-cycle we define $y(i)=x(i)$, so that in general $x(i)\le y(i)$ for all i, with equality in the case of a 1-cycle. Let $S(1)=\{1,...,n\}$, and for each i from 2 to $c+1$ let

$$S(i) = S(i-1) - \{x(i-1)\}'\{y(i-1)\}. \tag{4}$$

The smallest element in $S(i)$ is $x(i)$, and $y(i)$ can take any value in $S(i)$ including $x(i)$. The word $(g_1,...,g_c)$, where g_i is the number of elements of $S(i)$ which are smaller than $y(i)$, is the I-code for P, with the exception noted above. By padding each word on the right with zeroes to make it of length $n-1$ we obtain a lexicographical-order-preserving injection from the set of length-n involutions P into the set of $(n-1)$-tuples of non-negative integers $R(P)=(g_1,...,g_{n-1})$.

Example 6: If $P=(4,2,5,1,3,6)=(1,4)(2)(3,5)(6)$ in canonical cycle notation, then $x(1)=1$, $y(1)=4$, $x(2)=y(2)=2$, $x(3)=3$, $y(3)=5$ and $x(4)=y(4)=6$, so that $R(P)=(3,0,1,0,0)$ (the last zero is padding).

As P is changed to its lexicographical-order successor its pivotal element (which is now the *leftmost* letter which changes) may increase by more than 1, but the pivotal element of $R(P)$ always increases by 1 (see Table 3).

PUT TABLE 3 ABOUT HERE

Just as with FPFIs one can pass between a general involution P in one-line notation and $R(P)$ by successive computation of $S(i)$ from (4). In [11] the algorithms are written explicitly and it is stated that their complexity is $O(n^2)$. This estimate is sharp if it takes linear time to remove an element from an indexed set; as with FPFIs it can be reduced to $O(n \log n)$ or better using more efficient deletion algorithms.

In [11] there is an algorithm for passing from $R(P)$ to the lexicographical-order rank of P which takes $O(n \log n)$ arithmetic operations and uses a table of size $O(n^2)$. We improve the time estimate to $O(n)$ and the space estimate to $O(1)$.

We describe the set of words $(g_1,...,g_{n-1})$ that are the images of length-n involutions under the function R. In particular, given any such word with prefix $(g_1,...,g_{i-1})$, we find the maximum

value that g_i can attain. Let #(S) represent the cardinality of the set S. Then #(S(1))=n, and from (4) it follows that for all i, $1<i\le c+1$, if $g_{i-1}=0$ so that $y(i\text{-}1)=x(i\text{-}1)$, then #($S(i)$)=#($S(i\text{-}1)$)-1, and if $g_{i-1}>0$ so that $y(i\text{-}1)>x(i\text{-}1)$, then #($S(i)$)=#($S(i\text{-}1)$)-2. Since g_i is the number of elements of $S(i)$ less than $y(i)$, it can take any integer value from 0 up to #($S(i)$)-1, whch we denote by $b(i)$ (a function not only of i but also of the prefix $(g_1,...,g_{i-1})$), which can be calculated recursively from formula (5):

$$b(1) = n-1 \text{ and for } 1 < i \le c+1$$
$$b(i) = \begin{cases} b(i-1)-1 \text{ if } g_{i-1} = 0, \\ b(i-1)-2 \text{ if } g_{i-1} > 0. \end{cases} \tag{5}$$

From the definition of $b(i)$, $b(c+1)$=-1 because S(c+1)={}; so the values of all the $b(i)$ can be calculated from (5) for i decreasing as well as increasing.

Let $I(n)$ be the number of length-n involutions. There is a closed-form formula for $I(n)$ in [7], but it is easier to evaluate even a single value of $I(n)$ from the recursive formula (6) below:

$$I(0)=I(1)=1 \text{ and for } n\ge 2,\ I(n)=I(n\text{-}1)+(n\text{-}1)I(n\text{-}2). \tag{6}$$

This recursion follows from the fact that there are $I(n\text{-}1)$ involutions $P=(p_1,p_2,...,p_n)$ with $p_1=1$ and $I(n\text{-}2)$ involutions for each value 2,...,n of p_1 [11].

Let #$(g_1,...,g_{i-1})$ be the number of involutions whose padded I-codes have prefix $(g_1,...,g_{i-1})$. These involutions are induced by the involutions of the set $S(i)$, and since #($S(i)$)=$b(i)$+1, where $b(i)$ is computed recursively by (5), we have

$$\#(g_1,...,g_{i-1}) = I(b(i)+1). \tag{7}$$

To calculate the rank of $(g_1,...,g_{n-1})$ we note that among all the words with prefix $(g_1,...,g_{i-1})$, the number of words that come before the first word with prefix $(g_1,...,g_i)$ is 0 if g_i=0, and otherwise it is #$(g_1,...,g_{i-1},0)$+$(g_i\text{-}1)$#$(g_1,...,g_i)$, which, by (7) and (5), is $I(b(i))+(g_i\text{-}1)I(b(i)\text{-}1)$. We thus obtain the following ranking formula:

$$rank(g_1,\ldots,g_{n-1}) = \sum_{i=1}^{n-1} \begin{cases} 0 \text{ if } g_i = 0 \\ I(b(i)) + (g_i-1)I(b(i)-1) \text{ otherwise}, \end{cases} \tag{8}$$

where $I(n)$ is given by (6) and $b(i)$ by (5).

To find the word of rank r, for i from 1 to n-1 we let g_i be the largest value such that subtracting the summand of (8) from r leaves a non-negative number:

$$g_i = \begin{cases} 0 \text{ if } r < I(b(i)) \\ 1 + \lfloor (r - I(b)) / I(b-1) \rfloor \text{ otherwise}, \end{cases} \tag{9}$$

where $I(n)$ is given by (6) and $b(i)$ by (5).

Ranking and unranking the words can be done in $O(n)$ arithmetic operations with or without a precomputed table of I, so that the complexity of ranking and unranking involutions is dominated by that of passing between an involution and its I-code.

We define the following Gray code on the set of words $(g_1,...,g_{n-1})$ such that $0 \le g_i \le b(i)$: in any interval of words with prefix $g_1 g_2 ... g_{i-1}$, let g_i take the sequence of distinct values

$$s_i = \begin{cases} 1,2,..,b(i),0 \text{ if } g_1 + ... + g_{i-1} \text{ is even}, \\ 0,b(i),...,2,1 \text{ if } g_1 + ... + g_{i-1} \text{ is odd}. \end{cases} \tag{10}$$

The rank of the word $(g_1,...,g_{n-1})$ in Gray code order is given by

$$\sum_{i=1}^{n-1} \begin{cases} I(b(i)-1) * \begin{cases} g_i - 1 \text{ if } g_i > 0 \\ b(i) \text{ if } g_i = 0 \end{cases} \text{ if } g_1 + ... + g_{i-1} \text{ is even} \\ \begin{cases} 0 \text{ if } g_i = 0 \\ I(b(i)) + (b(i) - g_i) I(b(i)-1) \text{ otherwise} \end{cases} \text{ otherwise}. \end{cases} \tag{11}$$

To unrank, subtract the maximum value possible from r without making it negative by setting

$$g_i = \begin{cases} \begin{cases} 0 \text{ if } r \ge I(b(i)-1) * b(i) \\ 1 + \lfloor r / I(b(i)-1) \rfloor \text{ otherwise} \end{cases} \text{ if } g_1 + ... + g_{n-1} \text{ is even} \\ \begin{cases} 0 \text{ if } r < I(b(i)) \\ b - \lfloor (r - I(b(i))) / I(b(i)-1) \rfloor \text{ otherwise} \end{cases} \text{ otherwise}. \end{cases} \tag{12}$$

The first word $(g_1,...,g_{n-1})$ in this Gray code is (1,0,...,0) and the last one is (0,...,0). To find the successor to a given word we use the generic sequencing algorithm listed as Algorithm 1 with left and right reversed and with some optimizations introduced for the sake of efficiency.

To avoid computing the parity of $g_1+...+g_{i-1}$ from scratch for each i, we store the parity of $g_1+...+g_{i-1}$ in a Boolean variable *Odd*, and for each word we initialize another variable *OddPrefix* to *Odd* and then while scanning the word from right to left to search for the pivot we change *OddPrefix* whenever g_i is odd. To avoid computing $b(i)$ recursively (from left to right) from (5), we store c, begin the right-to-left scan from $i=c$ instead of n-1 (if $c=n$ we know that the word is (0,...,0)), and during the scan we initialize a variable b to -1 and increase it by 1 if $g_i=0$ and by 2 otherwise (expressing $b(i)$ in terms of $b(i+1)$ using (5)).

Further economies can be made using the following two theorems.

Theorem 4: If *Odd* is true, then the pivot is c if $g_c>0$ and c-1 otherwise. If *Odd* is false, let i be the largest index such that $g_i>0$ (if no such i exists then the pivot is 0). Then the pivot is i if $g_i>1$ and i-1 otherwise.■

Theorem 5: Let $i>0$ be the pivot. Then the suffix $(g_{i+1},g_{i+2},...,g_{n-1})$ changes from 00...0 to 10...0 if g_i jumps from an even number to 0, from 10...0 to 00...0 if g_i jumps from 0 to an even number, and is unchanged otherwise. The value of c decreases by 1 if g_i jumps from 0 to an odd number, increases by 1 if g_i jumps from an odd number to 0, and is unchanged otherwise. The value of *Odd* always changes.■

The reader is invited to verify these theorems for $n\leq 5$, as well as Theorem 6 which shows that this Gray code induces a Gray code on the set of length-n involutions, by examining Table 4.

PUT TABLE 4 ABOUT HERE

Theorem 6: In passing from an involution P and $R(P)=(g_1,...,g_{n-1})$ to its successor, one pair of elements of P swaps places if g_i passes between 0 and an odd number, two pairs swap places if g_i passes between 0 and an even number greater than 2 or between 1 and 2 if $g_{i+1}=1$, and three elements rotate under all other conditions.

Proof: We do a case-by-case analysis, illustrating each case with a table in which the first line contains the elements of S_i in increasing order, and the second and third lines contain the old and new elements (with the old names) of P occupying the positions whose index is given directly above them in the first line (see Table 5).■

PUT TABLE 5 ABOUT HERE

Implementation details of the sequencing algorithm for the involution P, with and without storing the auxiliary array $R(P)$, are given in [17]. If the auxiliary array is not stored, then only $O(1)$ auxiliary variables are used; in either case the worst-case time-complexity is $O(n)$. Storing the array reduced the execution time by about 4% and using Theorem 4 to find the pivot reduced execution time by about 20%.

The Gray code for I-codes can be sequenced in $O(1)$ time by storing the positions of all the non-zero elements on a stack. If i is the pivot of an I-code, then $x(i)$ is the pivot of the corresponding involution (see Table 5); however, the array $(x(1),x(2),...,x(c))$ can change by $\Omega(n)$ letters from one involution to the next. Worst-case constant-time sequencing of involutions is still an open problem.

Acknowledgment: R. Cori posed the problem of finding a Gray code for fixed-point-free involutions and suggested the original ranking and unranking algorithms. F. Ruskey sent me the article [4] which makes ranking and unranking asymptotically more efficient. P. Leroux suggested finding a Gray code for general involutions. And NSERC collaborated with the Computer Science Department of the Université du Québec à Montréal in financing this research.

REFERENCES

[1]: J.R. Bitner, G. Ehrlich and E.M. Reingold, *Efficient generation of the binary reflected Gray code and its applications*, Comm. ACM 19 (1976), pp. 517-521.

[2]: P.J. Chase, *Combination generation and graylex ordering*, in Proceedings of the 18th Manitoba Conference on Numerical Mathematics and Computing, Winnipeg, 1988, Congressus Numerantium 69 (1989), pp. 215-242.

[3]: R. Cori, *Un code pour les cartes planaires et ses applications*, Astérisque, Paris, 1975.

[4]: P.F. Dietz, *Optimal algorithms for list indexing and subset rank*, in Proceedings of the 1989 Workshop on Algorithms and Data Structures in Ottawa, Canada, published in Lecture Notes in Computer Science 382, Springer-Verlag, Berlin, 1989, pp. 39-46.

[5]: P. Eades and B. McKay, *An algorithm for generating subsets to fixed size with a strong minimal change property*, Information Processing Letters 19 (1984), pp. 131-133.

[6]: G. Ehrlich, *Loopless algorithms for generating permutations, combinations, and other combinatorial configurations*, J. ACM 20 (1973), pp. 500-513.

[7]: E. Foundas and A. Sapounakis, *Algorithms on involutions*, Journal of Information and Optimization Sciences 12 (1991), pp. 339-361.

[8]: P. Klingsberg, *A Gray code for compositions*, Journal of Algorithms 3 (1982), pp. 41-44.

[9]: D.E. Knuth, *The art of computer programming*, Vol. 3 (sorting and searching), Addison-Wesley, Reading, Mass., 1973.

[10]: C.N. Liu and D.T. Tang, Algorithm 452, *Enumerating M out of N objects*, Comm. ACM 16 (1973), p. 485.

[11]: D. Roelants van Baronaigien, *Constant time generation of involutions*, Congressus Numerantium 90 (1992), pp. 87-96.

[12]: F. Ruskey, *Adjacent interchange generation of combinations*, Journal of Algorithms 9 (1988), pp. 162-180.

[13]: F. Ruskey, *Simple combinatorial Gray codes constructed by reversing sublists*, in 4th International Symposium on Algorithms and Computation, published in Lecture notes in Computer Science 672 (1993), pp. 201-208.

[14]: C. Savage, *Gray code sequences of partitions*, Journal of Algorithms 10 (1989), pp. 577-595.

[15]: T.R. Walsh, *A simple sequencing and ranking method that works on almost all Gray codes,* Research report 243, Department of Mathematics and Computer Science, Université du Québec à Montréal, 1995, 53 pages.

[16]: T.R. Walsh, *A Gray code for involutions with a given number of fixed points*, Research report 10, Department of Computer Science, Université du Québec à Montréal, July 1997, 13 pages.

[17]: T.R. Walsh, *A Gray code for all length-n involutions*, Research report 9, Department of Computer Science, Université du Québec à Montréal, July 1997, 11 pages.

[18]: H. S. Wilf, *Combinatorial algorithms: an update*, SIAM, Philadelphia, 1989.

[19]: S.G. Williamson, *Combinatorics for computer science*, Computer Science Press, Rockville, 1985.

lexicographical order		Gray code order	
F	R(F)	F	R(F)
214365	00	214365	00
215634	01	341265	10
216543	02	432165	20
341265	10	532614	30
351624	11	632541	40
361542	12	645231	41
432165	20	546213	31
456123	21	456123	21
465132	22	351624	11
532614	30	215634	01
546213	31	216543	02
564312	32	361542	12
632541	40	465132	22
645231	41	564312	32
654321	42	654321	42

Table 1
The fixed-point-free involutions F of {1,...,6} and their I-codes $R(F)$ in lexicographical and Gray-code order.

Search for the leftmost g_i which is not at the last value in its sequence;
if such a g_i is found **then** { i is the *pivot* }
    Change g_i to the next value in the sequence;
    Change each g_j, $j<i$, which is not already at the first value in its sequence to the first value
else the current word is the last one.

Algorithm 1

A generic sequencing algorithm for transforming the word $(g_1,...,g_{n-1})$ into its successor using the sequence of distinct values attained by g_i as a function of its suffix $(g_{i+1},...,g_{n-1})$.

{Before execution:}
{for all j, e[j]=j unless (g[j],...,g[k-1]) is a z-word for some k>j, in which case e[j]=k. }
i:=e[1];
if i=n **then** { (g[1],...,g[n-1])=(z[1],...,z[n-1]) }
Done:=**True**; **return**
end if; {Otherwise (g[1],...,g[i-1])=(z[1],...,z[i-1]) but g[i]≠z[i] so that i is the pivot .}
{ In either case (e[1],...,e[i-1],e[i])=(i,2,3,...,i-1,i). }
(g[1],...,g[i-1],g[i]) := (a[1],...,a[i-1],h[i]); { O(1) changes in the case of a Gray code }
update any other auxiliary variables;
e[1]:=1;
if g[i]=z[i] **then** { Either e[i+1]=i+1, g[i+1]≠z[i+1] and (z[i]) is a z-word or else }
{ e[i+1]=k+1>i+1, (g[i+1],...,g[k]) was a z-word and now (g[i],g[i+1],...,g[k]) is a z-word. }
e[i]:=e[i+1]; { Now (g[i],...,g[k]) is a z-word and e[i]=k+1 whether k=i or not. }
e[i+1]:=i+1 { Since g[i]=z[i], g[i+1] cannot begin a z-word. }
end if. { Otherwise either i=1 or g[1]=a[1] and in either case g[1]≠z[1] and e[1] is still 1. }
{After execution:}
{for all j, e[j]=j unless (g[j],...,g[k-1]) is a z-word for some k>j, in which case e[j]=k. }

Algorithm 2

Generic sequencing algorithm for (g[1],...,g[n-1]) with loop-free pivot-finding using the auxiliary array (e[1],...,e[n]). Here a[i], z[i] and h[i] are, respectively, the first value, the last value and the next value after g[i] in the sequence of distinct values taken by g[i] in the interval of words with the suffix (g[i+1],...,g[n-1]). Initally g[j]=a[j] and e[j]=j for all j.

```
i:=e[1]; { i is the pivot in R(F)=(g[1],...,g[n-1]) }
if i=n then { F is the last FPFI }
   Done:=True; return
end if;
{ in what follows, x = x(i), y = y(i), g = g[i], and Up is true if g[i] is increasing }
if Odd then { R(F) has odd sum }
   x:=2*i-1; y:=f[x]; g:=y-2*i; Up:=(g is odd)
else
   x:=i; y:=f[x]; g:=y-(i+1); Up:=(g is even)
end if;
if (Up) then { g[i] must increase }
    g:=g+1; j:=y+1; { f[j] is the right neighbour of f[y]=x }
else  { g[i] must decrease }
    g:=g-1; j:=y-1; { f[j] is the left neighbour of f[y]=x }
end if;
swap f[y]=x with f[j]; swap f[x]=y with f[f[j]]=j;
Odd:=not(Odd);
e[1]:=1;
if (g=0) or (g=2*(n-i)) then { the new g[i] is at its last value }
   e[i]:=e[i+1];
   e[i+1]:=i+1
end if.
```

Algorithm 3

O(1)-time sequencing algorithm for the FPFI F=(f[1],...,f[2n])
using only the size-n auxiliary array e (initialized to (1,2,...,n)).

123$\underline{4}$	123$\underline{4}$
12$\underline{4}$5	12$\underline{3}$5
$\underline{2}$345	12$\underline{4}$5
13$\underline{4}$5	$\underline{1}$345
123$\underline{5}$	234$\underline{5}$
12$\underline{5}$6	$\underline{2}$346
$\underline{2}$356	13$\underline{4}$6
1$\underline{3}$56	12$\underline{4}$6
$\underline{3}$456	12$\underline{3}$6
$\underline{2}$456	12$\underline{5}$6
14$\underline{5}$6	$\underline{1}$356
12$\underline{4}$6	2$\underline{3}$56
$\underline{2}$346	$\underline{2}$456
13$\underline{4}$6	$\underline{1}$456
1236_	3456_

Table 2
The Liu-Tang Gray code (left) and the Eades-McKay Gray code (right) for the 4-combinations of {1,...,6}. The pivotal element is underlined.

```
r:=run(0);
if r>1 then { run(1)=r-1 and the sequence of distinct values taken by c[1] is given in Theorem 2 }
   if c[1] is not at its last value then { the pivot is 1 }
      c[1] := its next value; return
   end if
end if;
s:=run(r);  { the sequence of distinct values taken by c[r] is given in Theorem 2 }
if c[r] is not at its last value then { the pivot is r }
   c[r] := its next value; return
end if;
if r<k then { the pivot is r+1 }
   c[r+1] := c[r]+1
else { the current combination is the last one }
   Done := True
end if.
```

Algorithm 4

Non-recursive sequencing algorithm for the Eades-McKay Gray code for k-combinations of $\{1,\ldots,n\}$. Initially c[i]=i for all i and Done is False.

n=1	n=2	n=3	n=4	n=5	
R(P) P	R(P) P	R(P) P	R(P) P	R(P) P	R(P) P
1	0 12	00 123	000 1234	0000 12345	1000 21345
	1 21	01 132	001 1243	0001 12354	101o 21354
		10 213	010 1324	0010 12435	110o 21435
		20 321	020 1423	0020 12543	120o 21543
			100 2134	0100 13245	200o 32145
			11o 2143	011o 13254	201o 32154
			200 3214	0200 14325	210o 34125
			21o 3412	021o 14523	220o 35142
			300 4231	0300 15342	300o 42315
			31o 4321	031o 15432	301o 42513
					310o 43215
					320o 45312
					400o 52341
					401o 52431
					410o 53241
					420o 54321

Table 3

Involutions P of length n and the padded I-code $R(P)$ for $1 \le n \le 5$ in lexicographical order. A padding 0 is written as o and the pivotal element is underlined.

n=1	n=2	n=3	n=4	n=5	
R(P) P	R(P) P	R(P) P	R(P) P	R(P) P	R(P) P
1	1 21	10 213	100 2134	1000 21345	0100 13245
	0 12	20 321	11o 2*14*3	101o 21354	011o 13*25*4
		01 132	21o 3412	120o 21543	021o 14523
		00 123	200 3214	110o 2*14*35	0200 14325
			300 4231	210o 34125	0300 15342
			31o 4321	220o 35142	031o 15432
			010 1324	201o 32154	0010 12435
			020 1423	2000 32145	0020 12543
			001 1243	300o 42315	0001 12354
			000 1234	301o 42513	0000 12345
				320o 45312	
				310o 43215	
				410o 53241	
				420o 54321	
				401o 52431	
				4000 5*23*41	

Table 4

Involutions P of length n and the padded I-code $R(P)$ for $1 \le n \le 5$ in Gray code order. In $R(P)$, a padding 0 is written as o and the pivotal element is underlined. In P, two elements which swap or three which rotate are underlined, and if two pairs exchange then one of them is in italics.

Case 1: $g_i \geq 0$ and increases by 1: $y(i)$ rises to the next value in S_i, say j.

Subcase 1a: g_i increases from 1 to 2 and $g_{i+1}=1$: $p_j \neq j$

```
index:          x(i) y(i)  j    p_j
before:         y(i) x(i)  p_j   j
after:          j     p_j  x(i) y(i)   Two pairs swap.
```

Subcase 1b: otherwise: $p_j=j$.

```
index:          x(i)  *    *    *    y(i) p_j=j
before:         y(i)                  x(i)  j
after:           j                    y(i) x(i)   Three elements rotate.
```

Case 2: $g_i \geq 1$ decreases by 1: $y(i)$ falls to the next value in S_i, say j.

The analysis is similar to that of Case 1 except that the second and third columns (with $y(i)$ and j in the first line) change places.

Case 3: g_i jumps from $b(i)$ to 0: $y(i)$ drops from the largest member of S_i to $x(i)$.

Subcase 3a: $b(i)$ is even: the suffix $b(i)$00...0 changes to 010...0.

Subsubcase 3a.i: $b(i)>2$:

```
index:    x(i)  x(i+1)=y(i+1)  x(i+2)=y(i+2) *  x(i)
before:   y(i)      y(i+1)         y(i+2)       x(i)
after:    x(i)      y(i+2)         y(i+1)       y(i)  Two pairs swap.
```

Subsubcase 3a.ii: $b(i)=2$:

```
index:    x(i)  x(i+1)=y(i+1)  y(i)
before:   y(i)      y(i+1)     x(i)
after:    x(i)      y(i)      y(i+1)   Three elements rotate.
```

Subcase 3b: $b(i)$ is odd:

```
index:    x(i) *    *    y(i)
before:   y(i)          x(i)
after:    x(i)          y(i)  One pair of elements swaps.
```

Case 4: g_i jumps from 0 to $b(i)$: $y(i)$ jumps from $x(i)$ to j, the largest element of S_i.

Subcase 4a: $b(i)$ is even: the suffix 010...0 changes to $b(i)$10...0.

Subsubcase 4a.i: $b(i)>2$:

```
index:    x(i)=y(i) x(i+1)    y(i+1)    *    p_j=j
before:     y(i)    y(i+1)    x(i+1)         p_j=j
after:       j      x(i+1)    y(i+1)         y(i)  Two pairs swap.
```

Subsubcase 4a.ii: $b(i)=2$:

```
index:    x(i)=y(i) x(i+1)=p_j y(i+1)=j
before:     y(i)       j         p_j
after:       j         p_j       y(i)  Three elements rotate.
```

Subcase 4b: $b(i)$ is odd:

```
index:    x(i)=y(i) *    *    p_j=j
before:     y(i)              j
after:       j                y(i)   One pair of elements swaps.
```

Table 5

Transformation induced on an involution P by passing from R(P) to its successor in the Gray code