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.

1. Introduction

A length-$n$ involution is a permutation $(p_1, p_2, ..., p_n)$ of $\{1, ..., n\}$ such that for each $i$, $1 \leq i \leq n$, either $p_i = i$ or else $p_j = j \neq i$ 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 FPF, 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 \(p_i = i\) and \(p_x = y\) swaps with \(p_j = j\). 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 \(\{1, \ldots, n\}\).

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 $(2n-1)\times(2n-3)\times\ldots\times3\times1$ length-$2n$ FPFIs in one-line notation onto the Cartesian product of integer intervals $[1,2n-1]\times[1,2n-3]\times\ldots\times[1,3]\times[1,1]$. 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 $[0,2n-2]\times[0,2n-4]\times\ldots\times[0,2]$. Let $(x(1),y(1))\ldots(x(n),y(n))$, where $x(i)<y(i)$ for all $i$ and $x(1)<\ldots<x(n)$, be the canonical cycle notation for an FPI. For each $i$, $x(i)$ is the smallest element in $\{1,\ldots,2n\}-\{x(1),y(1),\ldots,x(i-1),y(i-1)\}$ and there are $2n-2i+1$ elements in the set

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

(1)

from which to choose $y(i)$. The function $R$ maps the FPI $(x(1),y(1))\ldots(x(n),y(n))$ onto $(g_1,\ldots,g_{2n-1})$, 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 $(g_1,\ldots,g_{2n-1})$ is just the number $g_1\ldots g_{2n-1}$, where each $g_i$ is in radix $2n-2i+1$. 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 FPI $F$ 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 $O(n \log n/\log \log n)$ using a more sophisticated data structure [4].

**Example 2:** Let $F = (f_1,\ldots,f_n) = (7,4,6,2,8,3,1,5)$. 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 $(g_1,...,g_{n-1}), 0 \leq g_i \leq 2(n-i)$, and we define $f(2n)$ 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).

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 $f(2n)$ 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].

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 \( R(F)=(g_1,\ldots, g_{n-1}) \) 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 \( g_{i+1}+\ldots+g_{n-1} \) 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 \( g_1+\ldots+g_{n-1} \). We store a Boolean variable \( \text{Odd} \) which is true if \( g_1+\ldots+g_{n-1} \) is odd; \( \text{Odd} \) is initialized to False for the first word \((0,\ldots,0)\) 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 \( f(2n) \) without having to store \( R(F) \) as an auxiliary array we use the following theorem.

**Theorem 1:** Let \( F=(f_1,\ldots, f_n) \) and let \( i \) be the pivot of \( R(F) \). If \( \text{Odd} \) is True, then \( x(i)=2i-1 \), \( g_i=y(i)-2i \), \( F=(2,1,4,3,\ldots,2i-2,2i-3,y(i)>2i,\ldots) \) and \( i \) is the smallest value such that \( f_{2i-1}>2i \); otherwise \( x(i)=i \), \( g_i=y(i)-(i+1) \), \( F=(2n,2n-1,\ldots,2n-i+2,y(i)<2n-i+1,\ldots,i-1,\ldots,2,1) \) 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 \( \text{Odd} \) is True, then \( g_i \) is the leftmost letter of \( R(F) \) which is not 0. By the definition of the function \( R \), \( x(1)=1 \), \( y(1)=2 \), \( x(2)=3 \), \( y(2)=4 \), \( \ldots \), \( x(i-1)=2i-3 \), \( y(i-1)=2i-2 \), \( x(i)=2i-1 \) but \( y(i)>2i \), leading to the first expression for \( F \) and thence to the first expression for \( i \). The set \( S(i)\setminus\{x(i)\} \) is the interval \([2i,2n]\), so that \( y(i)=g_i+2i \). If \( \text{Odd} \) 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 \( R \), \( x(1)=1 \), \( y(1)=2n \), \( x(2)=2 \), \( y(2)=2n-1 \), \( \ldots \), \( x(i-1)=i-1 \), \( y(i-1)=2n-1 \).
Afterwards, so that it always holds, the comments in Algorithm 2 imply that if it holds before the algorithm is executed then it will hold afterwards, so that $y(i)=g_i+i+1$.

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

**In example 3**, where $F=\{8,4,6,2,7,3,5,1\}$, $R(F)=\{6,4,1\}$ (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 $f_3=7$ 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 $O(1)$ auxiliary 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 $e=(e_1,\ldots,e_n)$ 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 $(g_{i+1},\ldots,g_{n-1})$. Since $g_i$ takes at least two distinct values, $a_i$ is never equal to $z_i$. A z-word is a subword $(g_j,\ldots,g_k)=(z_j,\ldots,z_k)$ 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 $(g_1,\ldots,g_{l-1})$ is a z-word, where $i\leq n$, then $i$ is the pivot. If $(g_1,\ldots,g_{n-1})=(z_1,\ldots,z_{n-1})$, 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 $j$, $e\neq j$ unless $(g_j,\ldots,g_{j+k-1})$ is a z-word for some $k\geq j$, in which case $e_k=j$. From this it follows that $e_1$ is always the pivot. The first word is $(a_1,\ldots,a_{n-1})$, and since $a_i\neq z_i$ for all $i$ there are no z-words. Since $(e_1,\ldots,e_n)$ is initialized to $(1,\ldots,n)$ 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 \( f(2n) \) is the list of all the length-2n FPFIs in the order given by the Gray code of Section 1 and we denote by \( f^R(2n) \) the same list in reverse order.

Let \( C=(c_1,c_2,\ldots,c_{2k}) \), where \( c_1<\ldots<c_{2k} \), be a 2\( k \)-combination (subset) of \( \{1,2,\ldots,n\} \) and \( F \) be an FPFI \( (f_1,f_2,\ldots,f_{2k}) \). Denote by \( P(C,F) \) the involution whose 1-cycles are the elements of \( \{1,\ldots,n\} \) 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 \( \{1,\ldots,8\} \) 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 2\( k \)-combination \( C \), we denote by \( P(C,f(2k)) \) the list of involutions \( P(C,F) \) as \( F \) runs over \( f(2k) \), with a similar definition for \( P(C,f^R(2k)) \). Given a list \( c(n,2k)=(C_0,C_1,\ldots) \) of 2\( k \)-combinations of \( \{1,\ldots,n\} \), we denote by \( p(c(n,2k),f(2k)) \) the list

\[
P(C_0,f(2k))\circ P(C_1,f^R(2k))\circ P(C_2,f(2k))\circ\ldots,
\]

where \( \circ \) 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 \( P(C_{2a},f(2k)) \) or \( P(C_{2a+1},f^R(2k)) \) 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 \( p(c(n,2k),f(2k)) \) 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 \( f(2k) \) 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 \( n=2k+1 \), \( C_a=(1,2,3,\ldots,2k) \), \( C_{a+1}=(2,3,\ldots,2k,2k+1) \) and \( F=(2,1,4,3,\ldots,2k,2k-1) \). Now \( C_{a+1} \) differs from \( C_a \) by the minimal amount - one element swapped out and one element swapped in - but \( P=(2,1,4,3,\ldots,2k,2k-1,2k+1) \) and \( P'=(1,3,2,5,4,\ldots,2k+1,2k) \) differ in all 2\( k \) elements because the element 1 swapped out of \( C_a \) and the element 2\( k \) swapped in have 2\( k \)-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 $p(n,k) = P(c(n,2k),f(2k))$ 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 $c(n,k)$ 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

$$c(n,k) = c(n-1,k) \circ c^R(n-2,k-1)n \circ c(n-2,k-2)(n-1)n,$$  

(2)

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

**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 $f(4)$. 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 $c(n,k)$ 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

\[ i,i+1,...,m-1,m \text{ if } k-i \text{ is even and } \text{run}(i) \text{ is even}, \]

\[ m,m-1,...,i+1,i \text{ if } k-i \text{ is odd and } \text{run}(i) \text{ is even}, \]

\[ m,i,i+1,...,m-1 \text{ if } k-i \text{ is even and } \text{run}(i) \text{ is odd}, \]

\[ m-1,...,i+1,i,m \text{ if } k-i \text{ is odd and } \text{run}(i) \text{ is odd}, \]

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

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

The sublists \(C(n-1,k)\), \(C^R(n-2,k-1)n\) and \(C(n-2,k-2)(n-1)n\) of \(C(n,k)\) 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 \(C(n,k)\) because \(C(n-1,k)\) consists of all the combinations such that \(c_k=n\), \(C^R(n-2,k-1)n\) of those such that \(c_k=n\) and \(c_{k-1}<n-1\), and \(C(n-2,k-2)(n-1)n\) 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 \(C(n-1,k)\) and the value \(n\) in each of the other two sublists; so it behaves as it should, since \(\text{run}(k)\) is always 0. The second-last letter \(c_{k-1}\) behaves as it should in the sublist \(C(n-1,k)\) 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 \(C^R(n-2,k-1)n\) it takes the sequence of values \(n-2,...,k,k-1\) by the induction hypothesis, and in \(C(n-2,k-2)(n-1)n\) it takes the value \(n-1\); so that in \(C^R(n-2,k-1)n\) \(o\) \(C(n-2,k-2)(n-1)n\) it takes the sequence of values \(n-2,...,k,k-1,n-1\), which satisfies the proposition since \(\text{run}(k-1)\) 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 \(C(n-1,k)\) 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 \(C^R(n-2,k-1)\) of what it is in \(C(n-2,k-1)\). But when \(n\) is appended at the right
of each word in \( c^R(n-2,k-1) \) to make \( c^R(n-2,k-1)n \), 1 is added to each \( k \), changing the parity of \( k-i \), and \( \text{run}(i) \) is never changed, which, according to the proposition, reverses the sequence of values attained by \( c_i \). It follows that the proposition holds in \( c^R(n-2,k-1)n \).

Appending \( n-1 \) and then \( n \) to the right of each word in \( c(n-2,k-2) \) changes \( k \) by 2 and \( \text{run}(i) \) by either 0 or 2; so the proposition holds in \( c(n-2,k-2)(n-1)n \) 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 \( r=\text{run}(0) \). 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 \( \text{run}(1)=r-1 \) must have the same parity or else \( m=1 \). For each \( i \), \( 1\leq i \leq r-1 \), \( c_i=c_{i+1}-1 \), so that \( c_i \) is also at its maximum value. If \( k-1 \) and \( \text{run}(1) \) have the same parity, then so do \( k-i \) and \( \text{run}(i)=r-i \), 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 \( r=\text{run}(0) \), 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 \( k-r \) and \( s=\text{run}(r) \) must have opposite parity.

Suppose that \( s \) is even, so that \( k-r \) is odd, and so \( c_r=r \). Then \( k-(r+1) \) is even and \( \text{run}(r+1)=s-1 \), which is odd. Also, since \( \text{run}(r)>1 \), \( c_{r+1} \) 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 $\text{run}(r+1)=s-1$, 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 $\text{run}(r+1)$ is odd. But that would imply that $c_{r+1}=c_{r+2}-1$, which contradicts the fact that $\text{run}(r)=1$.

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 $\text{Done}$, which is true if the current combination is the last one. The first combination is $(1,2,...,k)$ and for this combination $\text{Done}$ is False.

PUT ALGORITHM 4 ABOUT HERE

Once $r=\text{run}(0)$ and $s=\text{run}(r)$ 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 $\binom{2k}{k}$. 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 \( c(n,k) \). 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 \( c(n,k) \) 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):

\[
\text{Rank}(c_1, c_2, ..., c_n) = \sum_{i=1}^{k} (-1)^{k-i} \binom{c_i + 1}{i} \text{ if } \text{run}(i) \text{ is even} \\
\sum_{i=1}^{k} (-1)^{k-i} \binom{c_i}{i} \text{ if } c_i < c_i + 1 - 1 \\
\sum_{i=1}^{k} (-1)^{k-i} \binom{c_i}{i} \text{ otherwise}
\] -(k mod 2).  

(3)

The rank of the involution \( P(C,F) \) in \( P(C_0,f(2k))oP(C_1,f^R(2k))oP(C_2,f(2k))o... \) is \((2k-1)\times(2k-3)\times...\times3\) times the rank \( r \) of \( C \) in \( c(n,2k) = C_0,C_1,C_2,... \) plus the rank of \( F \) in either \( f(2k) \) if \( r \) is even or in \( f^R(2k) \) otherwise.

When the sequencing algorithm of Algorithm 3 processes the last word of \( f(2k) \), Odd will be False and \( e \) will be \((k,2,3,4,...,k)\). To begin generating \( f^R(2k) \) (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 \( P(C_0,f(2k))oP(C_1,f^R(2k))oP(C_2,f(2k))o... \) 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)≤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

\[ S(i) = S(i-1) - \{x(i-1)\} \cup \{y(i-1)\}. \]  

(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
We thus obtain the following ranking formula:

\[ \text{rank}(g_1, \ldots, g_{n-1}) = \sum_{i=1}^{n-1} \left\{ \begin{array}{ll} 0 & \text{if } g_i = 0 \\ I(b(i)) + (g_i - 1)I(b(i) - 1) & \text{otherwise} \end{array} \right. \]

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 \( i, 1<i\leq c+1 \), if \( g_{i-1}=0 \) so that \( y(i-1)=x(i-1) \), then \( #(S(i))=#(S(i-1)) \), and if \( g_{i-1}>0 \) so that \( y(i-1)>x(i-1) \), then \( #(S(i))=#(S(i-1))-2 \). 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 \( #(S(i)) \), which we denote by \( b(i) \) (a function not only of \( i \) but also of the prefix \( (g_1, \ldots, g_{i-1}) \)), which can be calculated recursively from formula (5):

\[
\begin{align*}
  b(1) &= n - 1 \text{ and for } 1 < i \leq c + 1 \\
  b(i) &= \begin{cases} 
  b(i-1) - 1 & \text{if } g_{i-1} = 0, \\
  b(i-1) - 2 & \text{if } g_{i-1} > 0. 
  \end{cases} \tag{5}
\end{align*}
\]

From the definition of \( b(i) \), \( b(c+1)=-1 \) because \( S(c+1)=\emptyset \); 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:

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

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

Let \( #(g_1, \ldots, g_{i-1}) \) be the number of involutions whose padded I-codes have prefix \( (g_1, \ldots, g_{i-1}) \). 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

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

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

\[
\text{rank}(g_1, \ldots, g_{n-1}) = \sum_{i=1}^{n-1} \left\{ \begin{array}{ll} 0 & \text{if } g_i = 0 \\
  I(b(i)) + (g_i - 1)I(b(i) - 1) & \text{otherwise} \end{array} \right. \tag{8}
\]
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:

\[
g_i = \begin{cases} 0 & \text{if } r < I(b(i)) \\ 1 + \left\lfloor \frac{(r - I(b))}{I(b - 1)} \right\rfloor & \text{otherwise} \end{cases},
\]

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 \( (g_1, \ldots, g_{n-1}) \) such that \( 0 \leq g_i \leq b(i) \): in any interval of words with prefix \( g_1g_2\ldots g_{i-1} \), let \( g_i \) take the sequence of distinct values

\[
s_i = \begin{cases} 1, 2, \ldots, b(i), 0 & \text{if } g_1 + \ldots + g_{i-1} \text{ is even,} \\ 0, b(i), 2, \ldots, 1 & \text{if } g_1 + \ldots + g_{i-1} \text{ is odd.} \end{cases}
\]

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

\[
\sum_{i=1}^{n-1} \left[ I(b(i)) - 1 \right] \begin{cases} g_i - 1 & \text{if } g_i > 0 \\ b(i) & \text{if } g_i = 0 \end{cases} \begin{cases} \text{if } g_1 + \ldots + g_{i-1} \text{ is even} \\ 0 & \text{if } g_i = 0 \\ I(b(i)) + (b(i) - g_i) I(b(i) - 1) & \text{otherwise.} \end{cases}
\]

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

\[
g_i = \begin{cases} 0 & \text{if } r \geq I(b(i)) \cdot b(i) \\ 1 + \left\lfloor \frac{r}{I(b(i)) - 1} \right\rfloor & \text{if } g_1 + \ldots + g_{n-1} \text{ is even} \\ 0 & \text{if } r < I(b(i)) \\ b - \left\lfloor \frac{(r - I(b(i)))}{I(b(i) - 1)} \right\rfloor & \text{otherwise.} \end{cases}
\]
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 \(i\geq 0\) 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.

**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). ■
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**


[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.


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

<table>
<thead>
<tr>
<th>$F$</th>
<th>$R(F)$</th>
<th>$F$</th>
<th>$R(F)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>214365</td>
<td>00</td>
<td>214365</td>
<td>00</td>
</tr>
<tr>
<td>215634</td>
<td>01</td>
<td>341265</td>
<td>10</td>
</tr>
<tr>
<td>216543</td>
<td>02</td>
<td>432165</td>
<td>20</td>
</tr>
<tr>
<td>341265</td>
<td>10</td>
<td>532614</td>
<td>30</td>
</tr>
<tr>
<td>351624</td>
<td>11</td>
<td>632541</td>
<td>40</td>
</tr>
<tr>
<td>361542</td>
<td>12</td>
<td>645231</td>
<td>41</td>
</tr>
<tr>
<td>432165</td>
<td>20</td>
<td>546213</td>
<td>31</td>
</tr>
<tr>
<td>456123</td>
<td>21</td>
<td>456123</td>
<td>21</td>
</tr>
<tr>
<td>465132</td>
<td>22</td>
<td>351624</td>
<td>11</td>
</tr>
<tr>
<td>532614</td>
<td>30</td>
<td>215634</td>
<td>01</td>
</tr>
<tr>
<td>546213</td>
<td>31</td>
<td>216543</td>
<td>02</td>
</tr>
<tr>
<td>564312</td>
<td>32</td>
<td>361542</td>
<td>12</td>
</tr>
<tr>
<td>632541</td>
<td>40</td>
<td>465132</td>
<td>22</td>
</tr>
<tr>
<td>645231</td>
<td>41</td>
<td>564312</td>
<td>32</td>
</tr>
<tr>
<td>654321</td>
<td>42</td>
<td>654321</td>
<td>42</td>
</tr>
</tbody>
</table>
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 \((g_1, \ldots, g_{n-1})\) into its successor using the sequence of distinct values attained by \( g_i \) as a function of its suffix \((g_{i+1}, \ldots, g_{n-1})\).
{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 \( e[j] = k \). \}

\( i := e[1] \);
\( \text{if } i = n \text{ then } \{ (g[1],...,g[n-1]) = (z[1],...,z[n-1]) \} \)

\text{Done:= True; return}
\( \text{end if;} \) \{Otherwise \( (g[1],...,g[i-1]) = (z[1],...,z[i-1]) \) but \( g[i] \neq z[i] \) so that \( i \) is the pivot.\}

\( (g[1],...,g[i-1],g[i]) := (a[1],...,a[i-1],h[i]) \); \{O(1) changes in the case of a Gray code\}
update any other auxiliary variables;
\( e[1] := 1 \);
\( \text{if } g[i] = z[i] \text{ then } \{ \text{Either } e[i+1] = i+1, g[i+1] \neq z[i+1] \text{ and } (z[i]) \text{ is a z-word or else} \}
\{ e[i+1] = k+1 > i+1, (g[i+1],...,g[k]) \text{ was a z-word and now } (g[i],g[i+1],...,g[k]) \text{ 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. \}
\( \text{end if.} \) \{Otherwise either \( i = 1 \) or \( g[1] = a[1] \) and in either case \( g[1] \neq z[1] \) 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 \( e[j] = k \). \}

\textbf{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 \((g[i+1],...,g[n-1])\). Initially \( g[j] = a[j] \) 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)).
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1234</td>
</tr>
<tr>
<td>1245</td>
<td>1235</td>
</tr>
<tr>
<td>2345</td>
<td>1245</td>
</tr>
<tr>
<td>1345</td>
<td>1345</td>
</tr>
<tr>
<td>1235</td>
<td>2345</td>
</tr>
<tr>
<td>1256</td>
<td>1234</td>
</tr>
<tr>
<td>2356</td>
<td>1346</td>
</tr>
<tr>
<td>1356</td>
<td>1246</td>
</tr>
<tr>
<td>3456</td>
<td>1236</td>
</tr>
<tr>
<td>2456</td>
<td>1256</td>
</tr>
<tr>
<td>1456</td>
<td>1356</td>
</tr>
<tr>
<td>1246</td>
<td>2356</td>
</tr>
<tr>
<td>2346</td>
<td>2456</td>
</tr>
<tr>
<td>1346</td>
<td>1456</td>
</tr>
<tr>
<td>1236</td>
<td>3456</td>
</tr>
</tbody>
</table>

**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,...,n}. Initially c[i]=i for all i and Done is False.
Table 3

<table>
<thead>
<tr>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
<th>n=4</th>
<th>n=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(P) P</td>
<td>R(P) P</td>
<td>R(P) P</td>
<td>R(P) P</td>
<td>R(P) P</td>
</tr>
<tr>
<td>1 0 12</td>
<td>00 123</td>
<td>000 12³4</td>
<td>0000 12³45</td>
<td>1000 21345</td>
</tr>
<tr>
<td>_1 21</td>
<td>01 132</td>
<td>001 1243</td>
<td>0001 12354</td>
<td>1010 21354</td>
</tr>
<tr>
<td>10 213</td>
<td>010 1324</td>
<td>0010 12435</td>
<td>1100 21435</td>
<td></td>
</tr>
<tr>
<td>_20 321</td>
<td>020 1423</td>
<td>0020 12543</td>
<td>1200 21543</td>
<td></td>
</tr>
<tr>
<td>_100 2134</td>
<td>0100 13245</td>
<td>2000 32145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_110 2143</td>
<td>0110 13254</td>
<td>2010 32154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200 3214</td>
<td>0200 14325</td>
<td>2100 34125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_210 3412</td>
<td>0210 14523</td>
<td>2200 35142</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300 4231</td>
<td>0300 15342</td>
<td>3000 42315</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_310 4321</td>
<td>0310 15432</td>
<td>3010 42513</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_310 4321</td>
<td>0310 15432</td>
<td>3010 42513</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_310 4321</td>
<td>0310 15432</td>
<td>3010 42513</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Involutions $P$ of length $n$ and the padded I-code $R(P)$ for $1 \leq n \leq 5$ in lexicographical order. A padding 0 is written as o and the pivotal element is underlined.
<table>
<thead>
<tr>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
<th>n=4</th>
<th>n=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(P)</td>
<td>P</td>
<td>R(P)</td>
<td>P</td>
<td>R(P)</td>
</tr>
<tr>
<td>1 1</td>
<td>1 21</td>
<td>10</td>
<td>213</td>
<td>100</td>
</tr>
<tr>
<td>_0 12</td>
<td>20</td>
<td>321</td>
<td>110</td>
<td>2143</td>
</tr>
<tr>
<td>01</td>
<td>132</td>
<td>210</td>
<td>3412</td>
<td>1200</td>
</tr>
<tr>
<td>_00</td>
<td>123</td>
<td>200</td>
<td>3214</td>
<td>1100</td>
</tr>
<tr>
<td>300</td>
<td>4231</td>
<td>210</td>
<td>34125</td>
<td>0300</td>
</tr>
<tr>
<td>310</td>
<td>4321</td>
<td>220</td>
<td>35142</td>
<td>0310</td>
</tr>
<tr>
<td>010</td>
<td>1324</td>
<td>2010</td>
<td>32154</td>
<td>0010</td>
</tr>
<tr>
<td>_020</td>
<td>1423</td>
<td>2000</td>
<td>32145</td>
<td>0020</td>
</tr>
<tr>
<td>001</td>
<td>1243</td>
<td>3000</td>
<td>42315</td>
<td>0001</td>
</tr>
<tr>
<td>_000</td>
<td>1234</td>
<td>3010</td>
<td>42513</td>
<td>_0000</td>
</tr>
<tr>
<td>3200</td>
<td>45312</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3100</td>
<td>43215</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4100</td>
<td>53241</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4200</td>
<td>54321</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4010</td>
<td>52431</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>52341</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4**

Involutions $P$ of length $n$ and the padded I-code $R(P)$ for $1 \leq n \leq 5$ 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 $g_{i+1} = 1$: $p_j \neq j$

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i)$</td>
<td>$y(i)$</td>
<td>$j$</td>
</tr>
<tr>
<td>$p_j$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i)$</td>
<td>$*$</td>
<td>$*$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$p_j$</td>
<td>$j$</td>
</tr>
</tbody>
</table>

Subcase 2: $g_i > 1$ 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 $S_j$ to $x(i)$.

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

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i)$</td>
<td>$x(i+1) = y(i+1)$</td>
<td>$x(i+2) = y(i+2)$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$y(i+1)$</td>
<td>$y(i+2)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i)$</td>
<td>$y(i)$</td>
<td>$y(i+1)$</td>
</tr>
<tr>
<td>$x(i+1)$</td>
<td>$y(i+1)$</td>
<td>$y(i)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i)$</td>
<td>$*$</td>
<td>$y(i)$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$x(i)$</td>
<td></td>
</tr>
<tr>
<td>$x(i)$</td>
<td>$y(i)$</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i) = y(i)$</td>
<td>$x(i+1)$</td>
<td>$y(i+1)$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$y(i+1)$</td>
<td>$x(i+1)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i) = y(i)$</td>
<td>$x(i+1) = p_j$</td>
<td>$y(i+1) = j$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$j$</td>
<td>$p_j$</td>
</tr>
<tr>
<td>$j$</td>
<td>$p_j$</td>
<td>$y(i)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(i) = y(i)$</td>
<td>$*$</td>
<td>$p_j = j$</td>
</tr>
<tr>
<td>$y(i)$</td>
<td>$j$</td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>$y(i)$</td>
<td>One pair of elements swaps.</td>
</tr>
</tbody>
</table>
Table 5
Transformation induced on an involution $P$ by passing from $R(P)$ to its successor in the Gray code