Generating Alternating Permutations Lexicographically

Bruce Bauslaugh and Frank Ruskey *

Department of Computer Science

University of Victoria, Victoria, B.C. V8W 2Y2, Canada

Abstract

A permutation $\pi_1\pi_2\cdots\pi_n$ is alternating if $\pi_1 < \pi_2 > \pi_3 < \pi_4 \cdots$. We present a constant average-time algorithm for generating all alternating permutations in lexicographic order. Ranking and unranking algorithms are also derived.

CR categories: F.2.2, G.2.1

1 Introduction

A permutation $\pi_1\pi_2\cdots\pi_n$ is alternating if $\pi_1 < \pi_2 > \pi_3 < \pi_4 \cdots$. Let $\mathbf{E}_n$ denote the set of alternating permutations of $\{1, 2, \ldots, n\}$, and let $E_n = |\mathbf{E}_n|$. The numbers E_n are called the Euler numbers. Some properties and applications of these numbers are discussed in Comtet [1], Entringer [2], Kemp [5], and Knuth and Buckholtz [7]. There are no published algorithms for generating alternating permutations, except for the algorithms of Ruskey [12] and Pruesse and Ruskey [10], which generate alternating permutations by transpositions, but only for the case where n is odd.

Many algorithms have been published for generating permutations. See the excellent survey of Sedgewick [14]. Lexicographic order is one of the most natural orders in which to generate sequences. Lexicographic generation algorithms have been given for permutations (Ord-Smith [9]), combinations (Nijenhuis and Wilf [8]), trees (Zaks and Richards [17], Zaks [16]), Young Tableau ([8]), and many other types of combinatorial objects. A desirable property of any algorithm for generating combinatorial

*Research supported by the Natural Sciences and Engineering Research Council of Canada under grant A3379.

Permutations			P-sequences		
$n=4$	$n=5$		$n=4$	$n=5$	
1324	13254	25143	121	1212	2412
1423	14253	25341	131	1312	2422
2314	14352	34152	221	1322	3312
2413	15243	34251	231	1412	3322
3412	15342	35142	331	1422	3412
	23154	35241		2212	3422
	24153	45132		2312	4412
	24351	45231		2322	4422

Figure 1: Lexicographic alternating Permutations for $n = 4, 5$, and the corresponding P-sequences.

objects is that it run in *constant average time*. This means that the total amount of computation done in generating all the objects, divided by the number of objects, is bounded by a constant. On the average and up to a constant factor no algorithm can run faster.

The inverses of alternating permutations may be thought of as the linear extensions of the so called "fence" poset (e.g., [10]), and thus may be generated by any of the various algorithms for generating all topological sortings. See, for example, Kalvin and Varol [4]. However none of these algorithms have constant average-time behavior. The algorithm presented here is simpler and more efficient.

Any algorithm for listing a set S of combinatorial objects imposes an ordering on the set of objects. Relative to the ordering, the *rank* of an object $s \in S$ is the number of objects that precede it in the list. Ranking algorithms are studied because they provide perfect hashing functions for the set S and since they often lead to interesting enumeration questions. An *unranking* algorithm takes an integer r, where $0 \leq r < |S|$, and returns the object having r as its rank. Unranking algorithms are useful for producing random elements of S, where each object is equally likely.

2 The Generation Algorithm

In this section we present the algorithm for generating alternating permutations lexicographically and an analysis showing that the algorithm runs in constant average-time. The lexicographically smallest alternating permutation is $1325476\cdots(n-1)(n-2)n$ if n is even and is $1325476\cdots n(n-1)$ if n is odd. The lexicographically largest alternating permutation is $(n-1)n(n-3)(n-2)\cdots 3412$ if n is even and is $(n-1)n(n-3)(n-2)\cdots 231$ if n is odd. Figure 1 shows lexicographic lists of alternating permutations for $n = 4, 5$.

Given a permutation π of $\{1, 2, \ldots, n\}$, define the *P-sequence* associated with π, $P[\pi] = p_1 p_2 \cdots p_{n-1}$, as follows. Element p_i is the number of elements π_j that satisfy

$\pi_j \leq \pi_i$ and $j \geq i$. In other words, p_i is one plus the number of elements that are less that π_i and to the right of it. For example, $P[351642] = 34132$. P-sequences are useful since they preserve lexicographic order, and free us from certain bookkeeping details in the algorithms. Permutation π is lexicographically less than π' if and only if $P[\pi]$ is lexicographically less than $P[\pi']$. A close variant of a P-sequence is the *inversion table*, $b_1 b_2 \cdots b_n$, of a permutation, where b_i is the number of elements to the left of i that are greater than i; see Knuth [6], pg. 12. Also related is the *inversion vector* defined in Reingold, Nievergelt, and Deo [11] by letting b_i be the number of elements greater than π_i and to its left. Thus an inversion table is an inversion vector indexed differently. Inversion vectors arise naturally in ranking permutations lexicographically ([11]). The proofs of the following lemma and corollary are elementary and left to the reader.

LEMMA 1: A necessary and sufficient condition for a P-sequence $p_1 p_2 \cdots p_{n-1}$ to be the P-sequence of an alternating permutation is that

$$p_1 \leq p_2 > p_3 \leq p_4 > p_5 \cdots \tag{1}$$

COROLLARY 1: If $p_1 p_2 \cdots p_{n-1}$ is the P-sequence of an alternating permutation and $p_0 = n$, then for $0 \leq l < n-1$,

$$1 \leq p_{l+1} \leq \min(n-l-1, p_l - 1) \qquad \text{if } l \text{ is even} \tag{2}$$

$$\max(2, p_l) \leq p_{l+1} \leq n-l \qquad \text{if } l \text{ is odd} \tag{3}$$

It is worth noting that any P-sequence must satisfy $1 \leq p_i \leq n-i+1$, so condition (2) only rules out $p_i = n-i+1$ and condition (3) only rules out $p_i = 1$. Subsequently, we assume that "P-sequence" means the P-sequence of an alternating permutation.

Corollary 1 leads us to the Pascal procedure `PAP` of Figure 2 for generating all P-sequences of alternating permutations in lexicographic order. The global integer array `p[0..n-1]` holds the P-sequence in `p[1..n-1]`. Given that $p_1 \cdots p_{l-1}$ is a prefix of a P-sequence, and $m + l = n$, the call `PAP`(m, k, l) generates all P-sequences of length $n-1$ with prefix $p_1 \cdots p_{l-1} k$. Thus `PAP`$(n, n, 0)$ generates all P-sequences (`p[0]` is ignored in this case), and `PAP`$(n-1, k, 1)$ generates all those for which $p_1 = k$. Local variables `lb` and `ub` are the upper and lower bounds on p_{l+1} as given in Corollary 1.

The algorithm is of the "backtracking with success at all leaves" variety. That is to say, the algorithm is recursive, and the leaves of the recursion tree correspond to the alternating permutations. The recursion tree for $n = 5$ is shown in Figure 4. Two recursion trees are shown; one for alternating permutations and the other for P-sequences. The *level* of the node in the tree is defined as usual, with the root at level zero. Parameter l is the level of a node. There are two kinds of levels in the tree; *even* levels and *odd* levels.

```
{Input: m+l = n, m > 0, p[1..l-1],k is a P-sequence prefix. }
{Output: all P-sequences with prefix p[1..l] where p[l] = k.}
procedure PAP ( m,k,l : integer );
var i,lb,ub : integer;
begin
   p[l] := k;
   if m = 1 then PrintIt
   else begin
      if odd(l) then begin
         lb := max( k, 2 );  ub := m
      end else begin
         lb := 1;  ub := min( k-1, m-1 )
      end;
      for i := lb to ub do PAP( m-1, i, l+1 )
   end
end {of PAP};
```

Figure 2: Pascal procedure for lexicographic P-sequence generation.

{ Input: $m + l = n$, $m > 0$, $\pi_1 \ldots \pi_{l-1} k$ is prefix of an alternating permutation. }
{ Output: all alternating permutations with prefix $\pi_1 \ldots \pi_{l-1} k$. }
proc AP (m, k, l : int ; S : intset);
var x : int; T : intset;
begin
    $\pi_l \leftarrow k$;
    if $m = 1$ **then begin**
        $\pi_n \leftarrow$ "last element of S";
        PrintIt
    end else begin;
        if odd(l) **then** $T \leftarrow \{x \in S \mid x < S_{max}$ and $x < k\}$
                else $T \leftarrow \{x \in S \mid x > S_{min}$ and $x > k\}$;
        forall $x \in T$ **do** AP($m - 1, x, l + 1, S - \{x\}$)
    end
end {of AP};

Figure 3: Recursive procedure to generate alternating permutations.

We now consider the changes to `PAP` to make it generate alternating permutations, rather than P-sequences. At a node at level l, a partial alternating permutation $\pi_1\pi_2\cdots\pi_l$ has been constructed. Let S denote the set $\{1,2,\ldots,n\} - \{\pi_1,\pi_2,\ldots,\pi_l\}$. Also, let S_{min} and S_{max} denote the minimum and maximum elements of S. Defining $\pi_0 = n+1$, at an odd level $l < n-1$,

$$\pi_{l+1} \in \{x \in S \mid S_{min} < x \text{ and } \pi_l < x\}$$

and at an even level $l < n-1$

$$\pi_{l+1} \in \{x \in S \mid x < S_{max} \text{ and } x < \pi_l\}$$

The reason that $\pi_{l+1} = S_{min}$ is excluded when $l < n-1$ is odd is that then there is no remaining value in S for π_{l+2} so that $\pi_{l+1} > \pi_{l+2}$. A similar remark applies in the even case with respect to S_{max}. At level n there is only one element in S and π_n gets the value of this element. This leads us to the algorithm of Figure 3, which has the same structure as `PAP`. Parameter m is the cardinality of S. To generate alternating permutations lexicographically, the elements of T must be selected by the **forall** statement in increasing order. With $S = \{1,2,\ldots,n\}$ the call AP$(n, n+1, 0, S)$ generates all alternating permutations (again, ignoring π_0), and with $S = \{1,2,\ldots,n\} - \{k\}$ the call AP$(n-1, k, 1, S)$ generates all those for which $\pi_1 = k$.

To efficiently implement AP, a global linked list in increasing order is used to maintain S. It is not necessary to build set T explicitly; we only need to know where in S to begin traversing. A Pascal implementation of AP may be obtained from the second author. The running time of either procedure `PAP` or AP is governed by the number of nodes in the recursion tree. We show in section 4 that the number of nodes in the recursion tree is $O(E_n)$.

3 Ranking and Unranking

In this section we develop ranking and unranking algorithms for alternating permutations. Two algorithms will be given: one for lexicographic order and one for another order that leads to a very simple ranking algorithm.

Let $\mathbf{E}(n,k)$ denote the set of elements π of $\mathbf{E}_n$ for which $\pi_1 = k$, and let $E(n,k)$ denote the number of elements in $\mathbf{E}(n,k)$. See Table 1. There is a simple recurrence relation for the $E(n,k)$ numbers as shown in the following lemma. This recurrence relation was discovered by Entringer [2] but the proof given here is different.

1 2 3 4

3 4 5 3 4 5 4 5 5

2 2 3 2 3 1 1 3 1 3 1 2 1 2 1 2

5 5 5 4 4 5 5 5 4 4 5 5 4 4 3 3

4 3 2 3 2 4 3 1 3 1 2 1 2 1 2 1

Alternating Permutations

1 2 3 4

2 3 4 2 3 4 3 4 4

1 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

P-sequences

Figure 4: Recursion trees for $n = 5$.

$n =$	1	2	3	4	5	6	7	8
$k = 1$	1							
2	1							
3	1	1						
4	2	2	1					
5	5	5	4	2				
6	16	16	14	10	5			
7	61	61	56	46	32	16		
8	272	272	256	224	178	122	61	
9	1385	1385	1324	1202	1024	800	544	272

Table 1: The $E(n,k)$ numbers.

LEMMA 2: If $n = 1$ then $E(n,k) = \delta_{k1}$. If $n > 1$ then if $k \geq n$ or $k < 1$ then $E(n,k) = 0$, and otherwise

$$E(n,k) = E(n,k+1) + E(n-1,n-k) \tag{4}$$

PROOF: Let $\pi \in \mathbf{E}(n,k)$. We classify such permutations according to whether $\pi_2 = k+1$ or not. If $\pi_2 \neq k+1$ then the elements k and $k+1$ may be swapped to obtain a permutation in $\mathbf{E}(n,k+1)$, and vice-versa. If $\pi_2 = k+1$ then define the permutation $\tau = \tau_1\tau_2\cdots\tau_{n-1}$ in $\mathbf{E}(n-1,n-k)$ where $\tau_j = n - \pi_{j+1}$ if $\pi_{j+1} < k$, and $\tau_j = n - \pi_{j+1} + 1$ if $\pi_{j+1} > k$, for $j = 1, 2, \ldots, n-1$. The permutation τ gives the corresponding permutation in $\mathbf{E}(n-1,n-k)$. For example, 36154728 in $\mathbf{E}(8,3)$ corresponds to 46153728 in $\mathbf{E}(8,4)$, and 34152768 in $\mathbf{E}(8,3)$ corresponds to 5746231 in $\mathbf{E}(7,5)$. ■

The proof of Lemma 2 provides an interesting way to rank alternating permutations. However, the ordering is not lexicographic. One might say that it is "pseudo-lexicographic", in the sense that it is lexicographic if in odd positions the ordering is increasing and in even positions the ordering is decreasing. The numbers $E(n,k)$ can be viewed as the number of paths in Figure 5 starting from a specific vertex and ending at the topmost vertex. With the topmost vertex at depth 2, the path starts at depth n. The number k is determined by the number of positions from the right or left following the directions of the edges. For example, the number of paths from the square vertex is $E(8,4)$, from the lower left vertex is $E(9,8)$, and from the lower right vertex is $E(9,1)$.

It is a simple matter, easily done by hand, to find the path corresponding to a permutation and to compute the rank from the path. In procedure AP let T_l denote the set T indexed by the level parameter l. The number of horizontal steps to follow is

$$|\{x \in T_{l-1} \,:\, x < \pi_l\}| \quad \text{if } l \text{ even},$$
$$|\{x \in T_{l-1} \,:\, x > \pi_l\}| \quad \text{if } l \text{ odd},$$

For example, consider the alternating permutation 46281537. Since $n = 8$ and $\pi_1 = 4$ we know where the path begins, e.g. at the square vertex. To determine the number of horizontal arrows to follow consider $T_1 = \{5,6,7,8\}$; since $\pi_2 = 6$ we take one horizontal step, and then one step up. Now consider $T_2 = \{5,3,2,1\}$; since $\pi_3 = 2$ we take two horizontal steps, and then one step up. Continuing in this manner we consider $T_3 = \{3,5,7,8\}$, $T_4 = \{5,3,1\}$, $T_5 = \{5,7\}$, $T_6 = \{3\}$, and $T_7 = \{7\}$, which results in the path of Figure 5(b). To rank this path we count the number of paths that precede it. This is the sum of the E numbers of the vertices that are "missed" on the way up. The E numbers corresponding to the missed vertices are shown explicitly in Figure 5(b). The rank is the sum of these numbers. For our example, the rank of 46281537 is (272+272+256)+(61+61)+(16)+(2)+(1). It should be apparent that the underlying algorithm is $O(n^2)$. Some savings can be attained

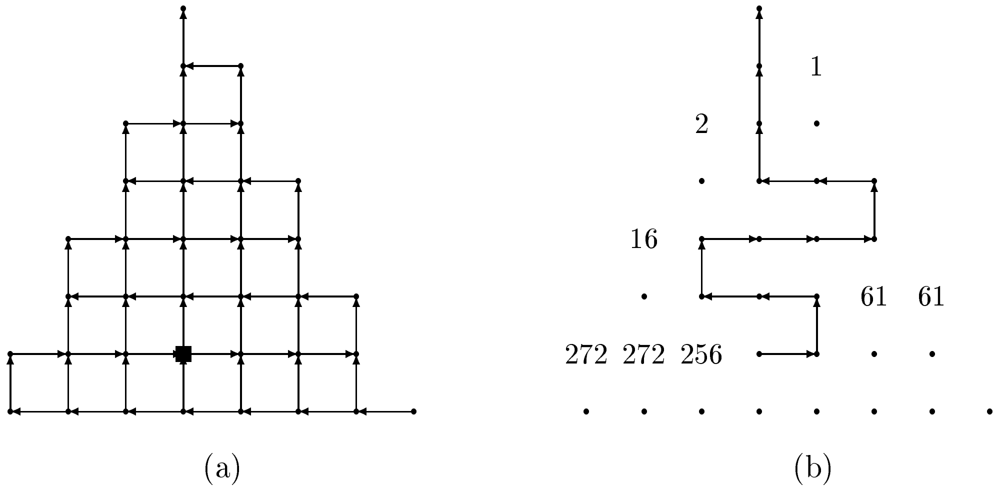

Figure 5: Ranking via Paths.

by realizing that the sums on a given depth are a single number on the next higher depth , but complications still occur in determining the number of horizontal steps to take.

We now derive a ranking algorithm for P-sequences of alternating permutations in lexicographic order. Fix a P-sequence $p_1p_2\cdots p_{n-1}$. Let $r(i)$ denote the number of P-sequences $q = q_1q_2\cdots q_{n-1}$ that lexicographically precede p and for which $p_1 = q_1, \ldots, p_{i-1} = q_{i-1}$ and $p_i \neq q_i$. The rank is $r(1) + r(2) + \cdots + r(n-1)$. The values of $r(i)$ can be computed from the following lemma.

LEMMA 3: For $1 \leq i \leq n-1$,

$$r(i) = \begin{cases} E(n-i+2, n-i+3-p_i) & \text{if } i \text{ odd} \\ E(n-i+2, p_{i-1}) - E(n-i+2, p_i) & \text{if } i \text{ even} \end{cases} \tag{5}$$

PROOF: First note that if $q_1q_2\cdots q_{n-1}$ is a P-sequence then so is $q_3\cdots q_{n-1}$. Assume that i is odd. We wish to know the number of P-sequences of the form $jq_{i+1}\cdots q_{n-1}$, where $j < p_i$. This is

$$\sum_{j=1}^{p_i-1} E(n-i+1, j) = E(n-i+2, n-i+3-p_i)$$

The equality follows by iterating (4). Now assume that i is even. We wish to know the number of P-sequences of the form $p_{i-1} j i q_{i+2} \cdots q_{n-1}$, where $j < p_i$. This is

$$\sum_{j=p_i}^{p_{i-1}-1} \sum_{i=1}^{j-1} E(n-i,i) = \sum_{j=p_i}^{p_{i-1}-1} E(n-i+1,n-j) = E(n-i+2,p_{i-1}) - E(n-i+2,p_i)$$

This finishes the proof. ■

Clearly, (5) provides a $O(n)$ algorithm for ranking P-sequences, if a table of the E numbers has been precomputed. To rank an alternating permutation, first compute its P-sequence.

It is also possible to derive an unranking algorithm that runs in time $O(n)$ and produces a P-sequence. The approach is standard and the details are left to the reader.

4 Analysis of the Generation Algorithm

The analysis of this section is similar to analyses found in Wilf [15], Ruskey and Proskurowski [13], and Hough and Ruskey [3]. Let $A(n,k)$ be the number of procedure calls in generating the elements of $\mathbf{E}(n,k)$. We assume that no call is made at the leaves of the tree. Referring to Figure 4, $A(5,1) = A(5,2) = 14$, $A(5,3) = 11$, and $A(5,4) = 6$.

Lemma 4: The A numbers satisfy $A(1,1) = 0$, $A(n,n-1) = 1 + A(n-1,1)$, and otherwise $A(n,k) = A(n,k+1) + A(n-1,n-k)$.

Proof: The number of calls to `PAP(n-1,k,l)` is $A(n,k)$ when `l` is odd. Let $B(n,k)$ denote the corresponding number of calls when `l` is even. Then $A(n,n) = B(n,1) = 0$ and otherwise for $1 \le k \le n$, from the program, we see that

$$A(n,k) = 1 + \sum_{j=k}^{n-1} B(n-1,j) \quad \text{and} \quad B(n,k) = 1 + \sum_{j=1}^{k-1} A(n-1,j)$$

From these equations it is easy to show that $A(n,k) = B(n,n-k)$, and then that the recurrence relation stated in the lemma is indeed valid. ■

Lemma 5: If $n > 0$ then $A(n,n-1) \le 3E(n,n-1)$, and if $1 \le k < n-1$ then $A(n,k) \le 3E(n,k) - 1$.

Proof: A simple induction using the recurrence relation of Lemma 4. ■

By Lemma 5 the algorithm does indeed run in constant average-time. Experimentally, for fixed k, the asymptotic ratio $A(n,k)/E(n,k)$ is slightly larger than 2.9.

5 Final Remarks

The computation of Euler numbers was considered by Buckholtz and Knuth [7]. They refer to E_n for n odd as the tangent numbers and for n even as the Euler numbers. The computation of these numbers is based on the recurrence relations

$$T_{n+1,k} = (k-1)T_{n,k-1} + (k+1)T_{n,k+1}$$

for the tangent numbers, and for the Euler numbers

$$E_{n+1,k} = kE_{n,k-1} + (k+1)E_{n,k+1}$$

These recurrence relations are clearly more complicated than (4). In particular we note that (4) does not require any multiplication, only additions. In [7] the numbers are given for $n \leq 120$ and have been calculated for $n \leq 835$ for the tangent numbers and for $n \leq 808$ for the Euler numbers. Using (4) and the UNIX utility BC we have calculated E_n for all $n < 1000$. The computation took about 5 hours on a Sun 3/50. This BC program may be obtained from the second author.

The efficiency of the ranking and unranking algorithms depends on the use of P-sequences rather than the permutations themselves. The process of converting a permutation to a P-sequence, or vice-versa, is easily accomplished with an $O(n^2)$ algorithm. By being somewhat more tricky (e.g. exercise 5.1.1.6 of [6]) this can be reduced to $O(n \log n)$.

Another order in which we might generate alternating permutations is *colex* order. This is lexicographic order in which the scanning is done from right to left. Because of the symmetry of alternating permutations the recursion trees for alternating permutations are isomorphic for either order. However, colex order of P-sequences and colex order of alternating permutations do not correspond. A colex algorithm for P-sequences can be developed that is similar to procedure `PAP` but avoids the need to compute maximums and minimums and so is somewhat more efficient. The corollary corresponding to Corollary 1, but for colex order is given below.

Corollary 2: If $p_1 p_2 \cdots p_{n-1}$ is the P-sequence of an alternating permutation and $p_n = 1$ then for $1 < l \leq n$,

$$1 \leq p_{l-1} \leq p_l \qquad \text{if } l \text{ is even} \tag{6}$$

$$p_l + 1 \leq p_{l-1} \leq n - l + 2 \qquad \text{if } l \text{ is odd} \tag{7}$$

6 Acknowledgement

The authors wish to thank one of the referees for their caustic remarks which lead to improvements in the presentation of section 2.

References

[1] L. Comtet. *Advanced Combinatorics.* D. Reidel, 1974.

[2] R.C. Entringer. A combinatorial interpretation of the Euler and Bernoulli numbers. *Nieuw Archief voor Wiskunde*, 14:241–246, 1966.

[3] T. Hough and F. Ruskey. An efficient implementation of the Eades, Hickey, Read adjacent interchange combination generation algorithm. *J. Combinat. Math. and Combinat. Computing*, 4:79–86, 1988.

[4] A.D. Kalvin and Y.L. Varol. On the generation of all topological sortings. *J. Algorithms*, 4:150–162, 1983.

[5] R. Kemp. *Fundamentals of the Average Case Analysis of Particular Algorithms.* Wiley-Teubner, 1984.

[6] D.E. Knuth. *Sorting and Searching.* Addison-Wesley, 1973.

[7] D.E. Knuth and T.J. Buckholtz. Computation of Tangent, Euler, and Bernoulli numbers. *Math. Computation*, 21:663–688, 1967.

[8] A. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms, 2nd Ed.* Academic Press, 1978.

[9] R.J. Ord-Smith. Generation of permutations in lexicographic order (algorithm 323). *Comm. ACM*, 2:117, 1968.

[10] G. Pruesse and F. Ruskey. Transposition generation of the linear extensions of certain posets. Technical Report DCS-91-IR, U. Victoria, 1988.

[11] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, 1977.

[12] F. Ruskey. Transposition generation of alternating permutations. Technical Report DCS-90-IR, U. Victoria, 1988.

[13] F. Ruskey and A. Proskurowski. Generating binary trees by transpositions. *J. Algorithms*, to appear.

[14] R. Sedgewick. Permutation generation methods. *Computing Surveys*, 9:137–164, 1977.

[15] H.S. Wilf. A unified setting for sequencing, ranking and selection algorithms for combinatorial objects. *Advances in Math.*, 24:281–291, 1977.

[16] S. Zaks. Lexicographic generation of ordered trees. *Theoretical Computer Science*, 10:63–82, 1980.

[17] S. Zaks and D. Richards. Generating trees and other combinatorial objects lexicographically. *SIAM J. Computing*, 8:73–81, 1979.