South African Computer Journal
Number 2
May 1990

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 2
Mei 1990

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Editorial

Information Systems Research: A Teleological Approach?

The request to write this editorial came at a very opportune time, coinciding as it did with an intense examination of the development of the field of information systems and an analysis of the progress of IS research. I have therefore used this opportunity to focus my thoughts and outline some of my conclusions. By doing so I don't pretend to answer any questions, merely perhaps to stimulate thought amongst those SACJ readers involved in IS research.

The last fifteen years has seen a tremendous growth in the study of information systems. During this period a number of journals devoted to IS research appeared such as *MIS Quarterly, The Journal of MIS, Information and Management* and *Data Base.* There are now many research-based activities: the International Conference on Information Systems; the annual IS doctoral dissertation colloquium; and various awards for IS research contributions. Hundreds of universities world-wide have formed information systems departments with (reasonably) standard curricula.

Yet with all this, what has *really* been achieved from a research viewpoint? Are we any closer to understanding the true nature of information systems? Is there a general unified theory of information systems? Is there even an accepted, unique body of IS knowledge? The answer to all of these must surely be no.

We have, I believe, achieved precious little. Yes, we do understand something of IS development approaches. We understand a little more now than we used to about how users interact with systems. But to get back to the first question, do we really understand what information systems *are* and how they work? No. Which begs the question: Why not?

There are, again I believe, a number of reasons, but the foremost must be that the majority of people in the IS research community either reside in the business schools of the USA or are drawn from other disciplines. These people, it would appear, are researching for research's sake; to publish in order to secure tenure or develop a research track record, not to further the body of knowledge of the subject. There seems an almost frantic zeal to generate and test hypotheses, trying to adopt and pursue what is seen to be a "scientific approach". But there is very little focus - there can't be, or the answers to my questions earlier would be yes rather than no!

Let me hasten to add that there is nothing unique about these IS researchers. "Publish or perish" is still very much alive and well! But also they are really not all that different from other social scientists. As Nagel [3] observed:

> "... in no area of social enquiry has a body of general laws been established, comparable with outstanding theories in the natural sciences in scope of explanatory power or in capacity to yield precise and reliable predictions ..."

Why should this be the case? Is it because the great intellects gravitate to the natural sciences and the social sciences pick up the second best who are incapable of generating these general laws? I hope not! The answer may well be that we have become locked into a particular research approach which is inappropriate to developing a body of social science, and more particularly, IS knowledge. Maybe we should be learning from our own source discipline (systems theory) and be developing a real research approach which complements our field of study.

To explore this further let me go back to the roots of information systems. What is an information system? Do we really have an accepted definition? Probably the most widely referenced is that provided by Davis and Olson [2]:

> "an integrated, user-machine system for providing information to support operations, management and decision-making functions in an organization. The system utilizes computer hardware and software; manual procedures; models for analysis, planning, control and decision making; and a database".

Note how this emphasizes the man-machine interrelationship and underscores computers as a core component when they are not even necessarily a part of the information system. The worst aspect is that it does little to describe what a system is, and this may well be one of the causes of our research dilemma. Again, if we draw on systems theory then a more appropriate definition might well be: "a hierarchical set of procedures utilizing information to monitor and control organizational performance". Note that this definition fits with general systems theory that *all systems* have four basic foundations: cybernetics, hierarchy, control and information [1].

An additional aspect not apparently recognised by IS researchers is that the information system, just like any other system, biological or otherwise, suffers from the problem first identified by our own Jan Christiaan Smuts [4]: that of holism. Simply put, this says that the whole is greater than the sum of the parts. This means that information systems, unlike science, cannot be reduced to simple isolated fields of enquiry and then analyzed or tested using hypotheses and laboratory experiments from which elaborate generalizations may be inferred. They have levels of complexity with new factors emerging at each level. The problem with most of the current research is that it starts out with a reductionist approach and then focuses on the highest (or lowest) level. Thus the majority of the topics have as their target the interaction between user and computer or the management or application of technology. There is very little research that is taking place at fundamental level, that of developing a general theory of information sytems. This is the teleological approach, searching for the natural laws and developing the theory based on deduction and logical development. Until we can advance *that* area of knowledge and, from a basis of these fundamental laws, develop a hierarchy of hypotheses that can *then* be tested, we will have little focus to our IS research. It will remain a fragmented, uncohesive smattering of the work of individuals who are merely grasping at tenure. There are few people who would today argue against the inclusion of information systems as a field of study at a university or as a fruitful research area. But until such time as we focus on the foundation theory, it will remain unstructured and immature.

References

[1] P Checkland, [1984], *Systems Thinking, Systems Practice*, John Wiley and Sons, New York.

[2] Davis and Olson, [1985], *Management Information Systems: Conceptual Foundations, Structure and Development*, 2nd Ed: MacGraw-Hill, New York.

[3] E Nagel, [1962], *The Structure of Science*, Routledge and Hegan Paul, London.

[4] J C Smuts, [1929], *Holism and Evolution*, MacMillan, London.

Prof Peter Lay
Assistant Editor: Information Systems

This SACJ issue is sponsored by
Department of Computer Science
University of Cape Town

On the Generation of Permutations

D Naccache de Paz
LITP, Université Pierre et Marie Curie (Paris VI), Aile 55-65, 4 Place Jussieu, 75252 Paris Cédex 05

Abstract

A new incremental algorithm for generating permutations on 1,2,..n (hereafter called "arrangements") is presented. The scheme generates arrangement k+1 by reversing a certain prefix of its predecessor and linking it to the remaining unchanged part of k.

The average size of this prefix tends very rapidly to 1 as n increases, and is equal to $\frac{37}{30}$ = 1,233... in the worst-case (where n=5).

The proposed algorithm uses only elementary CPU operations (namely register-shifts, additions and tests - no multiplications are made) which makes it suitable for low level language implementation and saves time. A final interesting property is the possibility of compressing the results into an array of 1!+2!+..+n! digits, instead of the nn! memory cells normally required for saving all the n! arrangements on n elements.

Keywords: *Algorithm, Permutation, Generation, Arrangement, Rank, Unrank*
Computing Review Categories: *G.2.1, F.2.2*

Received June 1989, Accepted October 1989.

1. Introduction

In computer science, the number of computational steps required to reach practically any valid result on generating permutations is extremely high for most methods. In fact, it is O(n!) where n is the number of elements of the basic set from which arrangements are to be generated. This order of complexity represents, in itself, a strong barrier to the generalized use of such methods. Although present-day computers are very fast and powerful, it is still necessary to perform a number of instructions and operations per generated permutation, which rapidly increase out of reach as n becomes large.

A second problem which, to the author's knowledge has not been discussed in previous work and to which a solution is proposed in section 4, is the compression of results into less than the nn! bytes that are normally required. The need to produce better methods for getting and saving values of such combinatorial objects has stimulated an analysis of the problem, resulting in the new approach to the computation which is presented here.

The first method for generating arrangements was presented by Tomkins [11]. This method, which is very lengthy, is essentially of historical value. In 1961 Wells [14] presented an algorithm which constituted an improvement of "about 20%" over the previous one.

Johnson's algorithm [5] is a method in which each arrangement is derived from its predecessor by a single interchange of two elements in adjacent positions. Tests comparing his method to the one presented in this article revealed a factor of improvement in computation times of at least 15 for $2 \le n \le 13$. On an IBM PC AT with an 80387 processor, Johnson's method (see [10] for the program) performs about 1000 arrangements per second (without displaying), while the average rate of our algorithm is 15500.

Plezcynski's algorithm [7] has quite simple bookkeeping, but does not show any outstanding improvement over other known methods. The most recent algorithm on our list ([15]), has a bigger ratio of operations per arrangement than [7] for large values of n (namely e versus e-1 respectively).

For additional literature about the subject, the reader may consult Trotter [12], Even [4], Dershowitz [1] and Ehrlich [2,3]. Also relevant are algorithms in certain sections of [9] (generation of subsets) and [8] (the travelling salesman problem).

We contend that it is possible to achieve the objective in less than $\frac{37}{30}n! \approx 1.23n!$ computational steps. The generating procedure is discussed in detail and presented (in the C language) in sections 2 and 3. The ranking and unranking algorithms are the subject of section 5, and in section 4 we give a time complexity estimation of our scheme.

2. The Generating Algorithm

The idea for this algorithm for generating arrangements comes from the robber and the digilock problem. (A digilock is an electronic door lock where a combination of k digits, a *digicode*, is required to allow access [13]). To simplify, assume that the code is a sequence of n digits, chosen without repetition from n digits - i.e. the code is some permutation of n digits. It is known that any digilock contains a register of n bytes where the input is stored. When a new digit arrives it is first moved to a buffer, then the digit in cell n-1 is shifted to cell n (covering the last value, which is lost), cell n-2 is

shifted to cell n-1, cell n-3 to cell n-2, etc. After shifting cell 1 to cell 2, the buffer content is moved to cell 1. The register is now scanned, and if the code is recognised, the order to open the door is sent.

Hence, the most intelligent approach for the thief is to try to generate all possible codes with a maximum number of overlays. Assume, for instance, that n=4. Then entering digits in the sequence 123412 automatically leads to successive tests of 1234, 2341 and 3412. The proposed algorithm links all the arrangements in one string of $\sum_{k=1}^{n} k!$ digits.

Throughout this paper, the following conventions will be used.

Convention 2.1 We denote by $W_n(k)$ the k^{th} arrangement (of n elements), and by $E_n(k)$, the size of the prefix to reverse in order to generate $W_n(k+1)$.

Convention 2.2 The vertical bar, "|" will represent the concatenation of two words. For instance: (678)|(12345) = (67812345).

Convention 2.3 Let $W = (a_1,a_2,..a_n)$ be a word. We will denote by $R^k[W]$ the word:
$(a_{k+1},a_{k+2},...a_n,a_k,a_{k-1},...a_1)$.

We will later require the following lemma:

Lemma 2.4 Let $\{W_n(1),...W_n(n!)\}$ be the n! distinct arrangements on the set $\{a_1,....a_n\}$, and let $\{V_{n+1}(1),...V_{n+1}(n!)\}$ be the set of arrangements on $\{a_1,...a_{n+1}\}$ defined by $V_{n+1}(k)=W_n(k)|a_{n+1}$. Then for all i and j such that $1\le i<j\le n!$ the arrangement $V_{n+1}(i)$ is an acyclic permutation of the arrangement $V_{n+1}(j)$.

As stated by Wells [14] ("...The bookkeeping of this generation scheme is handled, *as in most schemes of this type*, by an ordered set of indices t_k..."), most generation methods require a representation of the serial number of each arrangement by a sequence of positive integers (called the "factorial representation" in [7], t_k in [14], d_i in [5], etc.) Here we extrapolate from the work of Johnson [5] in order to derive such a representation of serial numbers. (By referring to section 3 of [5], and substituting the symbols N and d_i with t and r_{n-i}, the interested reader may verify the results below.)

Let t and n>1 be two integers such that $0\le t\le n!-1$. Then there exists a (unique) sequence of n-1 integers $r_0,r_1,...r_{n-2}$, (called the *factorial composition* of t on n!) such that $0\le r_i\le n-i-1$ and $t=\sum_{k=0}^{n-2} r_k \frac{n!}{(n-k)!}$. This sequence is defined by:

$t_{n-1} = t$

$t_i = t_{i+1} \bmod \frac{n!}{(n-i)!}$ for i=1,2..n-2; and

$r_i = t_{i+1} \operatorname{div} \frac{n!}{(n-i)!} \le n-i-1$,

where "div" stands for integer division.

Definition 2.5 On this basis, let us call *the translation function of* t *on* n the integer, $E_n(t)$, such that:
$r_{E_n(t)-1} \neq 0$ and $i<E_n(t)-1 \Rightarrow r_i=0$.

Given t and n, therefore, the translation function of t on n may be derived by recursively computing r_j for j=n-2,n-1,.. until the first nonzero value is obtained. $E_n(t)$ is then the index of this nonzero value + 1.

Two corollaries may be derived from definition 2.5:

Corollary 2.5.1 $1+E_n(t) = E_{n+1}(tn+t)$ and

Corollary 2.5.2 $E_n(n!-t) = E_n(t)$

The first corollary will be useful in the proof (by induction on n) of the scheme in which we will pass from a set of arrangements on n elements to a set of arrangements on n+1 elements. The interest of 2.5.2 is outlined in the program where, in practice, the computation of only n!/2 arrangements is necessary since if an arrangement is computed, then the reverse is known as well.

The next thing to do is to demonstrate that a set of words as defined in 2.6 contains all the arrangements on 1,2,..n.

Definition 2.6 Let us call the set, D_n, of words $\{W_n(1),...W_n(n!)\}$, such that:

2.6.1 $W_n(1)$ is an arrangement, and

2.6.2 $W_n(k+1) = R^{E_n(k)}[W_n(k)]$ for $1\le k\le n!-1$

an n-*digiset*.

The proof that an n-digiset contains all the arrangements on 1,2,..n proceeds in two steps, the first being the proof of the following lemma.

Lemma 2.7 Let D_n and D_{n+1} be an n-digiset and an (n+1)-digiset respectively, such that:

2.7.1 $W_n(1) = (a_1,..a_n)$ and

2.7.2 $W_{n+1}(1) = W_n(1)|a_{n+1} = (a_1....a_n,a_{n+1})$

Then $W_{n+1}(nk-n+k) = W_n(k)|a_{n+1}$ for $1\le k\le n!$

Proof (by induction on k):

It is given that $W_{n+1}(1) = W_n(1)|a_{n+1}$.

Assume that for some given index, k, we have: $W_{n+1}(nk-n+k) = W_n(k)|a_{n+1}$.

Consequently, since it can be shown from the structure of digitsets that: $W_{n+1}(nk-n+k+n) = a_{n+1}|W_n(k)$ or,

$$W_{n+1}(k(n+1)) = a_{n+1}|W_n(k) \qquad (2.7.3)$$

we have the following:

$W_{n+1}(k(n+1)+1)$

$= R^{E_{n+1}(k(n+1))}[W_{n+1}(k(n+1))]$ (per definition of R)

$= R^{E_{n+1}(k(n+1))}[a_{n+1}|W_n(k)]$ (from 2.7.3)

$= R^{E_n(k)+1}[a_{n+1}|W_n(k)]$ (from 2.5.1)

$= [R^{E_n(k)}[W_n(k)]]|a_{n+1}$

$= W_n(k+1)|a_{n+1}$ (per definition of R)

It is therefore true that:

$W_{n+1}(k(n+1)+1) = W_n(k+1)|a_{n+1}$ for k=1,2...n!
Since $k(n+1)+1 = n(k+1)-n+(k+1)$, the proof is achieved for k+1.

We now show that by starting with the arrangement $W_n(1) = (a_1,...a_n)$, and applying the sequence of prefix reversals as in 2.6.2, we generate all the n! arrangements, ending with $W_n(n!) = (a_n,...a_1)$. This is stated in the next theorem.

Theorem 2.8 The words of an n-digiset are all the arrangements on n elements.

Proof (by induction on n).

For n=2, the assertion holds, as (12) and (21) are all the arrangements on {1,2}.

Assuming that the words of the n-digiset, D_n, are all the arrangements on $\{a_1,...a_n\}$, we prove that the words of the (n+1)-digiset, D_{n+1}, are all the arrangements of $\{a_1,...a_{n+1}\}$.

By Lemma 2.7, we have:

$W_{n+1}(ni-n+i) = W_n(i)|a_{n+1}$ for i = 1,2,..n!

Hence, it follows that the set:

$\{W_{n+1}(1),W_{n+1}(n+2),..,W_{n+1}(nn!-n+n!)\}$ is a subset of D_{n+1} and (by 2.4) contains mutually disjoint acyclic arrangements.

Since y mod (n+1) $\neq$ 0 $\Rightarrow$ $E_{n+1}(y)=1$, and since $R^1[X]$ is a simple cyclic permutation of X, the proof is complete for n+1.

3 Implementation and Example

```
/* Generating Program */
#include <stdio.h>
#define out(i) putchar(w[(i+top)%n]); putchar(10)

main(argc,argv);
int argc;
char **argv;

{ /* MAIN */
    register copy,i,E,n,top=0;
    int r[80], w[80];
    if ((n=atoi(argv[1]))==1) {putchar(49); exit(0);}
    /* INITIALISATION OF VALUES */
    for (i=0; i<n; i++) {r[i]=0; w[i]=i+49;}
    /* MAIN LOOP: DONE n!/2 TIMES */
    while(!r[n-1])
      { /* WHILE */
        /* PRINTING w[i] & ITS REVERSED: o(n)
        */
        for (i=0; i<n; i++) out(i);
        for (i=n-1; i>-1; i--) out(i);
        E=1;
          /* PREFIX REVERSAL IN AVERAGE */
          /* CONSTANT TIME */
        if (++r[1]==n)
          { /* IF */
            while(r[E]==1+n-E){r[E]=0;r[++E]++;}
            for (i=0; i<E>>1; i++)
              { /* FOR */
                copy=w[(top+E-i-1)%n];
                w[(top+E-i-1)%n]=w[(top+i)%n];
                w[(top+i)%n]=copy;
              } /* FOR */
          } /* IF */
        top=(top+E)%n
      } /* WHILE */
} /* MAIN */
```

Example for n = 3

k=	$W_3(k)$=	$E_3(k)$
1	123	1
2	.231	1
3	..312	2
4	213	1
5	132	1
6	321	--
	123121321	

Example for n = 4

k=	$W_4(k)$=	$E_4(k)$
1	1234	1
2	.2341	1
3	..3412	1
4	...4123	2
5	2314	1
6	3142	1
7	1423	1
8	4231	2
9	3124	1
10	1243	1
11	2431	1
12	4312	3
13	2134	1
14	1342	1
15	3421	1
16	4213	2
17	1324	1
18	3241	1
19	2413	1
20	4132	2
21	3214	1
22	2143	1
23	1432	1
24	4321	--
	123412314231243121342132413214321	

4. Order of Complexity

Lemma 4.1 $\forall n, \sum_{k=1}^{n} k^2 k! = (n+1)!n - \sum_{k=1}^{n} k!$

This can also be written either as:
$\sum_{k=1}^{n-1} (n-k)kk!$ or as $\sum_{k=1}^{n} k! - n$ (see [6]).

It is clear that the total number of elements to move for generating all the arrangements on n elements is:
$\sum_{k=1}^{n!-1} E_n(k)$, and this value is (see 2.5):

$$\sum_{k=1}^{n-1} (n-k)kk! = \sum_{k=1}^{n} k! - n \leq \frac{37}{30}n!$$

The time complexity of our algorithm is thus about n! units of time, where a unit of time is the time spent by a processor in reversing a small string (whose average size is always less than $\frac{37}{30}$).

To get a visual idea of the relation between n and the average size (S_n) of the above mentioned prefix (which relates directly to the average number of operations per arrangement), the reader is referred to figure 1. It should be pointed out that I/O operations (O(n) per arrangement) are not included in the cost estimation.

Assume that the string of $\sum_{k=1}^{n} k!$ elements generated as in section 3 (namely (121) for n=2 elements, (123121321) for n=3 elements, (123412314231243312134213241321) for n=4 elements, etc.) was stored somewhere. Then one can re-obtain all the n! arrangements by shifting a "window" of size n on the above mentioned string. The (small) gain in operations arises from the fact that the reversal of the prefix is no longer necessary. (E must, however, be recomputed.) However, the resulting gain in memory is enormous (about 96.5% for n=30).

5. Ranking and Unranking Programs

The rank of an arrangement is an integer which indicates the order in which the arrangement was generated, i.e. k is the rank of arrangement $W_n(k)$. Ranking and unranking are the respective schemes for either computing an unknown rank of a given arrangement, or computing the value of an arrangement of a given rank. The algorithms for doing this are given directly below in C. However, since the proofs of the algorithms require the introduction of new concepts which are beyond the scope of this paper, they are not given here.

5.1 The Ranking Program

```
#include <stdio.h>
#include <string.h>
#define d(x) f[x]=strlen(w)-(strchr(w,48+x))-w)-1

unsigned int f[80];

unsigned int k(x,w0)
unsigned int x;
char w0;

{   /* FUNCTION K */
    return((x==2) ? ((unsigned int)(w0-48))
                  : (x*k(x-1,w0)-x+1f[x]));
}   /* FUNCTION K */
main(argc,argv)

int argc;
char **argv;
{   /* MAIN */
char w[80],n,h,*j;

if ((h=n=strlen(strcpy(w,argv[1])))==1)
    {putchar('1');exit(0);};
d(n);
for (;h>=3;h--)
{   /* FOR */
    *(j=strchr(w,48+h))=0;
    strcpy(w,strcat(j+1,w));
    d(h-1);
}   /* FOR */

printf("The rank is : %u",k(n,w[0]));
}   /* MAIN */
```

5.2 The Unranking Program

```
#include <stdio.h>
#include <string.h>
main()
{ /* MAIN */
    char w[88],d[80];
    int h,n,k;

    printf("n = "); scnaf("%d,&n);
    printf("k = "); scnaf("%d,&k);
    k--;

    for (h=n; h>=3; h--)
     { /* FOR h=n */
     d[h]=k%h;
     k/=h;
     } /* FOR h=n */
```

```
    if (!k) {strcpy(w,"12");} else {strcpy(w,"21");}
    for (h=3; h<=n; h++)
      { /* FOR h=3 */
      w[h-1]='0'+h;
      w[h]=0;
      strcpy(w,d[h]+strcat(w,w,d[h]));
      } /* FOR h=3 */
    printf("w = %s",w);
} /* MAIN */
```

6. Conclusion

In this paper we have presented a new efficient algorithm that generates all the n! arrangements on a given set in less than $\frac{37}{30}$n! units of time. A short review of the problem and currently available solutions were given. An example C-language program was given which implements the new algorithm. This program can easily be adapted to run as a module in an external main program.

Memory costs for the implementation are limited to two vectors of length n, and five simple "bookkeeping" variables. (The use of "copy" can even be avoided by using the trick: A-=B=-B+(A+=B); for directly transposing A and B.)

After presenting a time complexity estimate in section 4, it is shown how to find the position of a given arrangement, and how to construct the arrangement of a given rank.

References

[1] N Dershowitz, [1975], A simplified loop-free algorithm for generating permutations, *Bit*, 2(15), 158-64.

[2] G Ehlich, [1973], Loopless algorithms for generating permatations, combinations and other combinatorial configurations, *JACM*, 20(3), 500-513.

[3] G Ehlich, [1973], Algorithm 466 in "four combinatorial algorithms", *CACM*, 16(11), 690-691.

[4] S Even, [1973], *Algorithmic Combinatorics*, MacMillan, 2-11.

[5] S Johnson, [1963], Generation of permutations by adjacent transpositions, *Maths of Comp*, 17(83), 282-285.

[6] D Naccache de Paz, [1990], A note about $\Sigma k^m k!$, To appear in *The Pentagon*.

[7] S Plezcynski, [1975], On the generation of permutations, *IPL*, 3(6), 180-183.

[8] R Sedgewick, [1983], *Algorithms*, Addison-Wesley, 520-522.

[9] I Semba, [1984], An efficient algorithm for generating all k-subsets ($1 \le k \le m \le n$) of the set {1,...,n} in lexicographic order, *J. Algorithms*, 5(2), 281-283.

[10] D Stanton and S White, [1986], *Constructive combinatorics*, Springer-Verlag, 1-7 and 159-162.

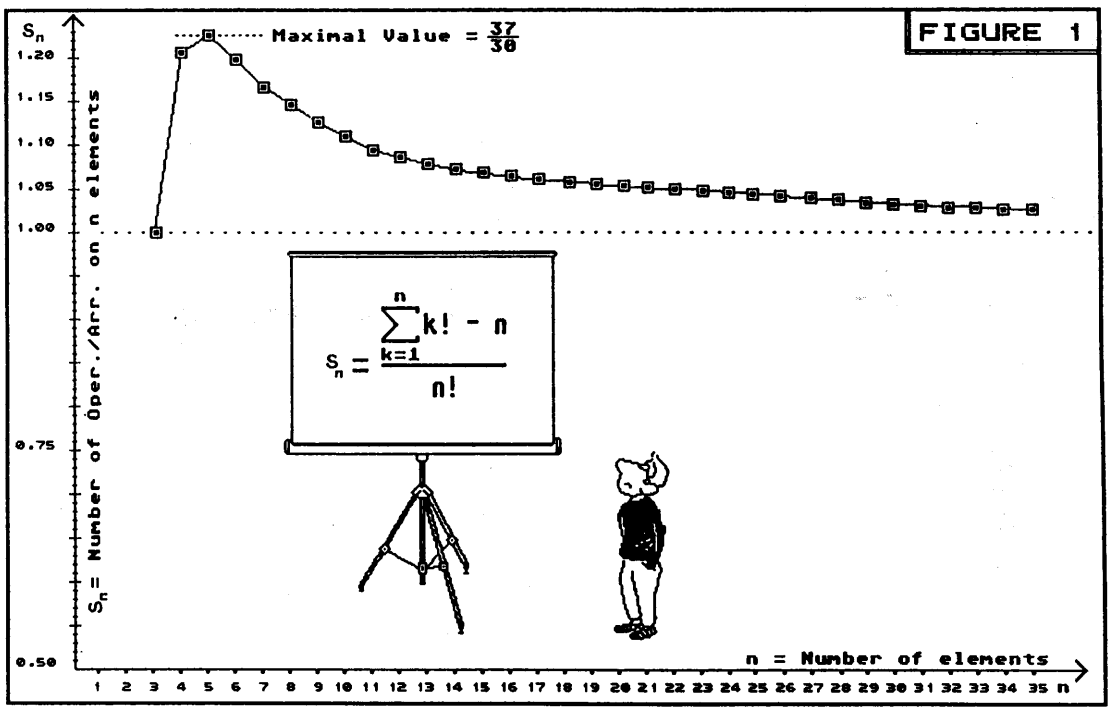
[11] C Tomkins, [1956], Machine attacks on problems whose variables are permutations, *Proceedings of symposium in Applied Maths*, Vol 6, Numerical Analysis, McGraw-Hill, 195-211.

[12] H Trotter, [1962], Algorithm 115 in Perm., *CACM*, 5(8), 434-435.

[13] N Vilenkin, [1971], *Combinatorics*, Academic Press, 4.

[14] M Wells, [1961], Generation of permutations by transposition, *Maths of Comp.*, 15(74), 192-195.

[15] S Zaks, [1984], A new algorithm for generation of permutations, *Bit*, 24(2), 196-204.



NOTES FOR CONTRIBUTORS

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between such cases as:
  - upper and lower case letters;
  - the letter O and zero;
  - the letter I and the number one; and
  - the letter K and kappa.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:
  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.
  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may provided in one of the following three formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies.

Charges

A charge per final page, scaled to reflect scanning, typesetting and reproduction costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format:	R80-00
ASCII format:	R60-00
Camera-ready format :	R20-00

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 2, May 1990
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 2, Mei 1990
ISSN 1015-7999

Contents