Discrete Mathematics and Theoretical Computer Science Proceedings **AA (DM-CCG)**, 2001, 165–176

On Minimal Strings Containing the Elements of S_n by Decimation

Robert Erra [1], Nik Lygeros[2], and Nigel Stewart [3]

[1]*ESIEA, Paris, France, erra@esiea.fr*
[2]*Inst. G. Desargues, Universit Lyon 1, Lyon, France, lygeros@desargues.univ-lyon1.fr*
[3]*RMIT University, Australia, nigels@eisa.net.au*

received January 30, 2001, *revised April 10, 2001*, *accepted April 16, 2001.*

The *permutations by decimation* problem is thought to be applicable to computer graphics, and raises interesting theoretical questions in combinatory theory. We present the results of some theoretical and practical investigation into this problem. We show that sequences of this form are $O(n^2)$ in length, but finding optimal solutions can be difficult.

Keywords: Hyperplane Arrangements, Symmetric Group, Permutations, q-analogs

1 Introduction

One of the most fundamentals objects of combinatorics is perhaps $\mathcal{S}_n$ – the set of all permutations of an alphabet of size n. This paper investigates a new problem concerning $\mathcal{S}_n$ - the strings containing $\mathcal{S}_n$ by *decimation*. Decimation means that each permutation is formed by deleting entries in the string. This investigation has focused on strings of minimal length satisfying the decimation constraint. These strings are refered to these as being *optimal* solutions. Optimal solutions are not necessarily unique – for example, the reversal of a optimal solution is also an optimal solution.

This problem is applicable to a computer graphics rendering. Abstract formulations will be presented, followed by some simple algorithms for generating non-optimal solutions. It will be shown that the optimal solution length is n^2-2n+4. Techniques for obtaining optimal solutions are investigated, including exhaustive trial-and-error and Genetic Algorithm (GA) evolution of solution strings. Variations such as alternating groups, and *r-permutations* of n are also discussed. Applications and related problems such as de Bruijn sequences are presented, followed by the results of computer searches.

2 Statement of the problem

2.1 Formalization of the problem

A_n is the set of n elements $\{a,b,\cdots,n\}$, and $\mathcal{S}_n$ the set of the $n!$ permutations of A_n. $\mathcal{L}(n)$ as the set of solution strings containing $\mathcal{S}_n$ by decimation. A solution is denoted L_m, where $L_m \in \mathcal{L}(n)$ and $m = |L_m|$. A string x is contained by decimation in y iff $y = \lambda\, x_1\, \lambda\, x_2\, \lambda\, \ldots\, \lambda\, x_{|x|}\, \lambda$, where x_i is the ith element of x, and $\lambda = (A_n)^*$. The problem is to determine the smallest m for which L_m exists.

2.2 DFA Formulation

The problem may also be formulated in terms of *deterministic finite automata* (DFA) from the field of Computer Science[16]. DFA's consist of finite sets of states and alphabet, and transition function for each input character for each state. The language of a machine M, denoted $L(M)$ are the inputs that leave M in a valid finshing state.

Figure 1 illustrates a DFA accepting all strings containing *abc* by decimation. The machine begins processing in the start state, denoted by the arrow at S_1. The accepting finish state is denoted by the double circle, at S_4. Six appropriately configured DFA's could be used to perform a decimation test for the six permutations of three objects. The problem is to determine the shortest input string accepted by all $n!$ DFA's, corresponding to each permutation of n objects. A related problem is the construction the simplest possible DFA accepting all permutations of n objects by decimation.

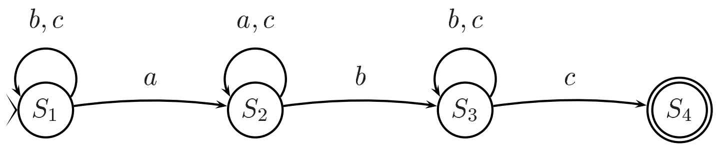

Fig. 1: State machine representation.

2.3 Optimal solution of simple cases

Denoting decimated elements with '$\star$', optimal solutions for $n = 2,3$:

- $n = 2$, $L_3 = aba$ since $\mathcal{S}_2$ are contained by decimation:

  1. $ab\star \to ab$
  2. $\star ba \to ba$

- $n = 3$, $L_7 = abcacba$ since $\mathcal{S}_3$ are contained by decimation:

  1. $abc\star\star\star\star \to abc$
  2. $a\star c\star\star b\star \to acb$
  3. $\star bca\star\star\star \to bca$
  4. $\star b\star ac\star\star \to bac$
  5. $\star\star ca\star b\star \to cab$
  6. $\star\star c\star\star ba \to cba$

Exhaustive search by means of computer programs have verified that no shorter solutions exist.

2.4 Some properties of $\mathcal{L}(n)$

Properties of $\mathcal{L}(n)$ and L_m provided here without proof:

- $|\mathcal{L}(n)| = \infty$, since $L_m \cdot x \in \mathcal{L}(n)$, where $x \in A_n$
- $|L_m| \geq n$
- $s_1 \cdot x \cdot x \cdot s_2 \in \mathcal{L}(n) \to s_1 \cdot x \cdot s_2 \in \mathcal{L}(n)$, where $x \in A_n$ and $s_1, s_2 \in {A_n}^*$

2.5 Solutions of length n^2

A solution may be formed by concatenating n copies of s, where $s \in \mathcal{S}_n$. Denoting '|' as the concatenation operator: $s|s|\cdots|s$. Since the length of s is n, the length of solutions in this form is n^2.

Solutions in this form can be shown to contain $\mathcal{S}_n$ by decimation. s contains all n characters, ensuring that the o'th copy of s will cater for the o'th character of any element of $\mathcal{S}_n$. This property is illustrated in Figure 2.

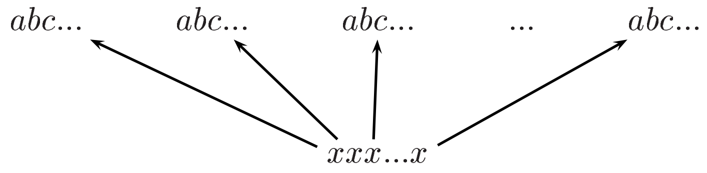

Fig. 2: Basis for solution sequence of length n^2.

2.6 Solutions of length $n^2 - n + 1$

The solution algorithm described previously may be further refined. Rather than concatenating n copies of *any* $s \in \mathcal{S}_n$, any sequence of elements of $\mathcal{S}_n$ may be used : $s_1|s_2|\cdots|s_n$. This generalization still ensures that $\mathcal{S}_n$ are contained by decimation, since each copy contains all n characters.

The flexibility allows exploitation of redundancy. Any repeated character in a solution can be regarded redundant, since no such sequence occurs in $\mathcal{S}_n$. Neighbouring blocks may be selected so that redundant entries are introduced.

One possible scheme is to select s_x and s_y so that concatenations $s_x|s_y$ or $s_y|s_x$ introduce redundancy. The first entry in s_x should be the same as the last entry in s_y, and vice versa. Examples for $n = 2, 3$:

- $n = 2$, $s_x = ab$ and $s_y = ba$:
  $s_x|s_y \to ab\ ba \to aba$
- $n = 3$, $s_x = abc$ and $s_y = cba$:
  $s_x|s_y|s_x \to abc\ cba\ abc \to abcbabc$

As illustrated in Figure 3, $n - 1$ entries may be removed, resulting in a solution length of $n^2 - n + 1$.

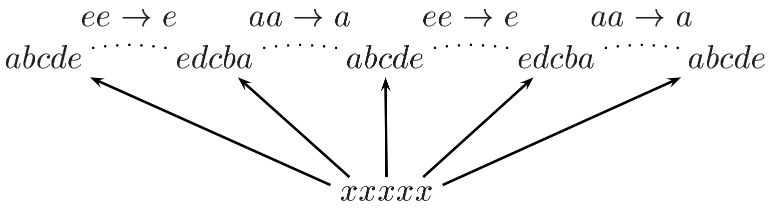

Fig. 3: Basis for solution sequence of length $n^2 - n + 1$.

3 Experimental results

3.1 Generate and test strategy

A brute-force means of finding optimal solutions is by generating every possible sequence of a particular length and testing whether it contains all permutations by decimation. The execution time of this algorithm is proportional to the size of the search space, and the number of decimation tests required for each candidate.

The size of the search space is n^l, where n is the number of objects, and l is the length of the candidate solutions. The maximum number of decimation tests is $n!$ - candidates are rejected as soon as a decimation test fails.

The following table illustrates the cost of exhaustive search for $n < 5$. The explosive growth of possibile candidates means that this approach is only useful for $n < 5$.

n	l	n^l	$n!$	$n^l n!$	CPU Time
1	1	1	1	1	~ 0 sec
2	3	8	2	16	0.04 sec
3	7	$2,187$	6	$13,122$	0.50 sec
4	12	$16.7M$	24	$403M$	27 sec

3.2 Testing normalized string space

It is possible to dramatically reduce the size of the search space by taking advantage of properties of optimal solutions. Two criteria are described here which reduce the search space without excluding any optimal solutions.

- *Normalised* strings are in the form
  $a\ \{a\}^*\ b\ \{a,b\}^*\ c\ \{a,b,c\}^*\ d\ \cdots$

  Normalised strings have the property that no element may appear until all 'lower' elements have appeared. At least one 'a' must appear somewhere to the left of the first 'b', and 'b' to the left of 'c', and so on. Any string can be normalised by defining an appropriate one-to-one relation. Similarly, any normalised string may be converted to $n!$ other strings via one-to-one relations. It is proposed that searching normalised string space is sufficient for finding optimal solutions, based on the observation that the normalisation of any solution is also a solution. This reduces the size of the search space by a factor of (roughly) $n!$.

- Optimal solutions do contain a repetition of an object. Candidates such as 'abba' do not need to included in the seach space.

Normalised with No Repeats (NNR) is defined as being the set of normalised strings containing no repetitions. NNR may be generated algorithmically as follows:

```
Inactive List = { a,b,c,... }
Active List = {}
Initial Sequence = {}

GenerateNNR(inactive,active,sequence)
{
  if sequence length is L,
    output sequence
  otherwise
    if inactive list length is non-empty
      pop  X from inactive list
      push X to active list
      GenerateNNR(inactive,active,sequence+X)
    if active list length is non-empty
      for each X in active list
        if X is not last element of sequence
          GenerateNNR(inactive,active,sequence+X)
}
```

The following table illustrates the size of the NNR search space in comparision to the naive search space. This provides a feasible search space for $n = 5$, an incremental improvement at best.

n	l	n^l	NNR	CPU Time
1	1	1	1	0.00 sec
2	3	8	1	0.00 sec
3	7	$2,187$	31	0.07 sec
4	12	$16.7M$	28,501	1.69 sec
5	19	$2 \cdot 10^{13}$	$3 \cdot 10^9$	18 hour

3.3 Genetic Algorithm approach

Based on an abstract model of biological evolution, the *Genetic Algorithm*[4][9] (GA) is a general adaptive search technique. A population of candidate solutions are evolved under selective pressure towards better solutions. This selective pressure often takes the form a *fitness function* (or *objective function*). Genetic operators such as sexual reproduction and mutation are applied to produce subsequent generations of candidates.

A GA for this problem has been implemented, using the *GAlibrary* C++ Genetic Algorithm Library by MIT. The GA is configured as follows:

- The fitness function defined as being the number of elements of S_n contained by decimation.
- The size of candidates is kept constant.
- The built-in genetic operators of crossover and mutation are used.
- Elitest selection strategy.

This approach has been found to useful for finding solutions for any n. Several issues were considered, regarding GA application to this problem:

- The discrete fitness function is a problem when many equally fit solutions exist in the population. It is not clear whether a continuous or less granular fitness function can be formulated. One idea would be to penalise longer-living candidates.
- It appears to be more efficient to evolve a large population of equally fit candidates than to breed more intensively from a smaller pool. This implies an elitest selection strategy that retains a percentile of candidates for the next generation.
- Some characterisation of the search space would be expected to improve the tuning of the GA search. It is not clear, for example, whether every candidate of a particular fitness is of equal 'distance' from being a solution. Also, it is not clear whether global maxima are located in the vicinity of local maxima.
- Careful tuning of the configuration can improve the rate of convergence of the GA. This is a painstaking process, and it seems that a meta-GA could be used for this purpose.

Some results produced by the GA are as follows:

n	length	P	G	s
2	3	??	??	aba
3	7	??	??	$abcabac$
4	12	200(?)	19	$abdcabdacdba$
5	19	500	57	$acbedacbeadcbaecdab$
6	28	1000	102	$badcefbdaecbfeadbcefabdacefb$

where P is the number of agents of the population at each generation, G is the number of generation needed to obtain the string s.

4 Generation of solutions

4.1 Optimal solutions

Theorem : A minimal normalized string containing by decimation the elements of $S_n$has the following pattern :$a\{n\}a\{n-1\}a...a\{n-1\}a\{n\}a$, where the gaps {n} and {n-1} are filled by repeating the string bcd...nbcd...n until is possible.

First minimal normalized strings are : a, aba, $abcabca$, $abcdabcadbca$, $abcdeabcdaebcadebca$.

Lemma : For $n < 4$ the minimal normalized strings for one, two and three letters are : a, aba and $abcabca$. The same pattern as those described by the theorem.

Proof: Exhaustive computation on the normalized strings.

Proof of the theorem by induction
For $n < 4$, we use the minimal strings of the lemma. For $n > 3$, we use the following algorithm to construct minimal strings : at first we take the string P_{n-1} then we put as a prefix the string $abcd \cdots n$ and after we put in the radical $n-3$ times the letter n just after the letter $n-1$.

1 letter , P_1 of length 1 :	a
2 letters, P_2 of length 3 :	aba
3 letters, P_3 of length 7 :	$abcabca$
4 letters, P_4 of length 12 :	$*****a*bcda**bcad**bca$
5 letters, P_5 of length 19 :	$******abcdea*bcdae*bcade*bca$
6 letters, P_6 of length 28 :	$abcdefabcdeafbcdaefbcadefbca$

All the P_k for $k < n$ are minimal strings containing by decimation the elements of S_k. Now we construct our $P_n$with the previous method and we consider the n different classes of permutations : $n*****, *n****$, $**n*******n**$, $****n*$, $*****n$. All the permutations of the first class are obviously contained by P_n because if we delete the letters n of the radical of P_n we obtain P_n, and we know that it is optimal. All the permutations of the second class are obviously contained by P_n because the prefix contains the $n-1$ letters and the remain contains obviously the $n-2$ letters of the permutation and this is also optimal by construction. For this two classes we used the first occurrence of the letter n which is at the end of the prefix. For the next classes $**n***$, ... if it is necessary we use the second occurrence of the letter n and so on. For the last class of permutation $*****n$ we use the last occurrence of the letter n because after this letter we have only bca which is replaced at the beginning by the prefix $abcd...n$ so this class is also contained by decimation in P_n.

So starting with P_{n-1} which is a minimal string we put the optimal prefix $abcd...n$ in the set of normalized strings and at the end we use $n-3$ times the letter n which is also optimal.

Corollary : For $n > 3$ we have $Card(P_n) = n^2 - 2n + 4$.

Proof : By construction for $n > 3$ we have :
$Card(P_n) = n + Card(P_{n-1}) + n - 3$ and $P_4 = 12$.

5 Origins, Applications and Related problems

5.1 The border crossing problem

A traveller intends to travel from Portugal to Germany via Spain and France. At each border crossing, they must present the appropriate paperwork to the guard, in order to be allowed to continue their journey. Unfortunately, the traveller can only understand English, and does not know which document should be offered to each guard. They are also too distracted to remember which document had been accepted at previous check-points.

The traveller holds three documents, for entry into Spain, France in Germany. What is the optimal sequence of presentation of documents in order to travel from Portugal, to Spain, to France, and finally Germany? The solution is SFSGSFS, a sequence which contains every permutation of the three documents by decimation.

5.2 Application to Computer Graphics

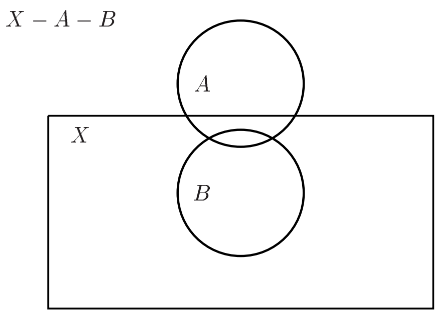

Fig. 4: Trickle Algorithm Surface Sorting.

Three dimensional computer graphics is generally concerned with the formation of an image relating to a three dimensional model. One stage of this process is determining the visible surface for each pixel on the screen. There are several known methods of solving this problem, which can be considered as kinds of sorting algorithms. Sometimes a complete ordering is required, while in others only the closest surface is required. The Z-buffer algorithm is one well known solution this problem.

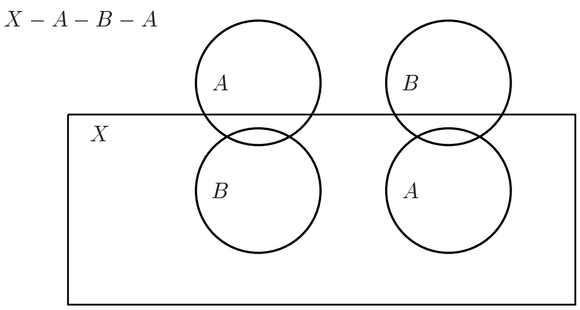

Fig. 5: CSG Algorithm based on sequence of subtractions.

A central problem in CSG and solid modelling is determining the surfaces satisfying volumetric constraints. This may be resolved either on an object or pixel basis. Image-space approaches use per-pixel classification of surfaces with respect to volumes to determine visible surfaces[15]. A solid modelling operation may be formulated in terms of volumetrically subtracting a number of convex volumes from a convex volume. The image is formed by volumetrically subtracting each volume in sequence. The

Trickle[2] algorithm sorts subtracted volumes from front to back, as illustrated in Figure 4. An alternative is to subtract in a sequence that caters for any ordering of volumes, from front to back, as in Figure 5. This sequence is equivalent to one containing S_n by decimation. In Figure 5, subtracting in the sequence *aba* ensures a correct result, regardless of the ordering of the volumes.

5.3 Alternating Groups

A variant of the problem is to find minimal strings containing $\mathcal{A}_n$ by decimation. $\mathcal{A}_n$ is the *alternating group*, the group of even or odd permutations. The reader is referred to Conway[1] and Kargapolov[8] for the definition and properties of $\mathcal{A}_n$. Computer programs have found the following solutions for the even alternating group:

n	length	solution	optimal
1	1	*a*	yes
2	2	*ab*	yes
3	5	*abcab*	yes
4	10	*abcdacbdac*	yes
5	17	*abcdeabcdaecbdaec*	yes
6	26	*acbdefcabedcfbadecbfdacebf*	unknown
7	38	*abdcefgbacedbfcagebdacfbgaedcbafgdecba*	unknown

5.4 Related Problems

The binary de Bruijn sequence is the sequence of 2^n bits $a_1a_2\cdots a_{2^n}$ such that each binary string of size n is somewhere in the string s, contiguously. These sequences have been called *Ouroborean Rings* by Ian Stewart[14]. The word Ourobouros comes from an ancient Egyptian representation of a snake swallowing its tail and is has a particular meaning in ancient alchemical works. Generalising this definition, we are concerned with n-ary sequences whose elements are drawn from the alphabet $\{0,1,\cdots,n-1\}$. A $n-ary$ de Bruijn sequence of span v is cyclic sequence of length n^v such that every possible v-tuples occurs precisely once in a period of the de Bruijn sequence, as a contiguous part. For example, a $2-ary$ de Bruijn sequence of span 2 is 010.

A problem that can be solved with Ouroborean Rings is the *Baltimore Hilton Inn* problem[3]. A cipher lock system uses a 4 digit code to allow access. Assuming that the code consists of digits from 0 to 9, there are 10^4 possible codes for the lock. An attacker may try every possible combination, requiring $4\cdot10^4$ keypresses, or $2\cdot10^4$ on average. But since the lock opens whenever the correct code is the last 4 digits, fewer keypresses are required. The best sequence of attack is the Ouroborean ring associated with the lock. (Length?)

These sequences are called *full length nonlinear shift register cycles* by Fredricksen[3]. The first solution was offered by Flye-Sainte Marie in the last century[10]. An important application of the Ouroborean rings is the generation of pseudo-random binary sequences of maximal length[6]. Other applications have a long history[14],[3].

6 Results

6.1 Length of Optimal Solutions

The length of the shortest known solution for each n is presented in the following table:

n	1	2	3	4	5	6	7	8	9
$\mathcal{S}_n$	1	3	7	12	19	28*	39*	52*	67*
$\mathcal{A}_n$	1	2	5	10	17	26*	38*		

6.2 r-permutation *Optimal Solutions*

A variation of the problem is to find strings containing *r-permutations* of A_n. That is, permutations of r selections from A_n. The following table contains the length of the shortest known solution for each n and r.

n	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2		3	5	7	9	11	13	15*	17*
3			7	10	13	16	19*	22*	25*
4				12	16	20*	24*	28*	32*
5					19	24*	29*	35*	40*
6						28*	35*	41*	
7							39*		
8								52*	
9									67*

6.3 Optimal Solutions

Solutions that are known to be optimal are listed here in normalised form.

n	l	solution
2	3	*aba*
3	7	*abcabac*
		abcabca
		abcacba
		abcbabc
		abcbacb
		abacaba
		abacbab

n	l	solution
4	12	*abcdabcadbac*
		abcdabcadbca
		abcdabcadcba
		abcdacbadbca
		abcdacbadcab
		abcdacbadcba
		abcdbacbdabc
		abcdbacbdacb
		abcdbacbdcab
5	19	*abcdeabcdaebcadbcea*
6	28	*abcdefabcdeafbcdaefbcadefbca*
7	39	*abcdefgabcdefagcdbeafgdcbaegdcfabcdefga*
8	52	*abcdefghabcdefgahcdfbeagcdfbhaefgdcbahedcfgabcdefgha*
9	67	*abcdefghiabcdefghaibcdefgahibcdefaghibcdeafghibcdaefghibcadefghibca*

7 Conclusion and Further Work

7.1 Conclusion

The problem of sequences containing permutations by decimation has been defined and investigated. The lower bound for the length of solutions has been found to be $n^2 - 2n + 4$ for $n > 3$. Some methods of searching for solutions by means of computer software have been investigated, including use of a genetic algorithm. Some optimal solutions for $1 \leq n \leq 9$ have been found. An algorithm for generating sub-optimal solutions of length $n^2 - n + 1$ has been presented.

7.2 Further Work

We believe that there is scope for further investigation of several issues in relation to the permutations by decimation problem.

Characterisation of the search space could improve methods of finding optimal solutions. The distribution or structure of solutions throughout the search-space is only vaguely understood. Also, it is not known how the number of objects affects the number of unique solutions, or the probability that a neighbourhood contains one.

The robustness and performance of the Genetic Algorithm search could certainly be improved, by considering features of the search space and optimising GA parameters. A meta-GA design, allowing dynamic optimisation of the search is one proposed improvement. There are many other variations to the manner in which the GA may be configured, few of which have been invesigated.

It is anticipated that application to the problem of CSG rendering [15],[5],[2] will be the major area of work resulting from this research.

7.3 Acknowledgments

This work has been supported by the Co-Operative Research Center for Intelligent Manufacturing Systems & Technologies.

References

[1] J. Conway et. al., "ATLAS of finite groups" Clarendon Press, Oxford, 1985

[2] D. Epstein, F. Jansen and J. Rossignac, Z-Buffer Rendering from CSG: The Trickle Algorithm, IBM Research Report RC 15182, Nov. 1989.

[3] H. Fredricksen, A survey of full length nonlinear shift register cycle algorithm, *SIAM Rev.*, **24** (1982), 195-221.

[4] D. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, 1989.

[5] J. Goldfeather, S. Molnar, G. Turk and H. Fuchs, Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning, *IEEE CG&A*, **9**, No. 3 (1989), 20-28.

[6] S. Golomb, "Shift register sequences", Holden-Day, San Fransisco, 1967.

[7] B. Heap, Permutations by interchanges, *Computer J.*, **6** (1963), 293-294.

[8] M. Kargapolov and I. Merzliakov, "Eléments de la théorie des groupes" Editions Mir, Moscou, 1985

[9] L. Kronsjö, D. Shumsheruddin, "Advances in Parallel Algorithms", Blackwell Scientific Publications, 1992.

[10] Flye-Sainte Marie, Solution to problem number 58, *L'Intermédiaire des Mathématiciens*, **1** (1894), 107-110.

[11] R. Sedgewick, Permutations generation methods, *Computing Surveys*, **9** (1977), 137-164.

[12] S. Skiena, "Implementing Discrete Mathematics (Combinatorics and Graph Theory with Mathematica)", Addison Wesley, 1991.

[13] S. Stein, "Mathematics, The Man-Made Universe", W.H. Freeman, San Fransisco, 1976.

[14] I. Stewart, "Game, Set and Math", Penguin Mathematics, 1989, published initially in *Scientific American*,

[15] N. Stewart, G. Leach and S. John, An Improved CSG Rendering Algorithm, *Proc. 1998 Eurographics/SigGraph Workshop on Graphics Hardware*, (1998), 25-30.

[16] T. Sudkamp, "Laguages and Machines", Addison Wesley, 1991.