International Journal of Computer Mathematics

ISSN: 0020-7160 (Print) 1029-0265 (Online) Journal homepage: www.tandfonline.com/journals/gcom20

Construction of skolem sequences

Sharaf. A. Eldin, N. Shalaby & F. Al-Thukair

To cite this article: Sharaf. A. Eldin, N. Shalaby & F. Al-Thukair (1998) Construction of skolem sequences, International Journal of Computer Mathematics, 70:2, 333-345, DOI: 10.1080/00207169808804756

To link to this article: https://doi.org/10.1080/00207169808804756

Published online: 19 Mar 2007.

Intern. J. Computer Math., Vol. 70, pp. 333–345
Reprints available directly from the publisher
Photocopying permitted by license only

CONSTRUCTION OF SKOLEM SEQUENCES

A. SHARAF ELDIN[a,*], N. SHALABY[b] and F. AL-THUKAIR[c]

[a] *Department of Information Systems, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia;*
[b] *Department of Math. and Stat., Memorial University of Newfoundland, St. John's, NF, Canada A1C 5S7;*
[c] *Department of Mathematics, College of Science, King Saud University, Riyadh, Saudi Arabia*

(Received 1 December 1997; In final form 28 February 1998)

In this paper, we use a hill-climbing algorithm to construct Skolem sequences of arbitrary order. The details of the algorithm is given. Samples of the generated sequences are also given.

Keywords: Skolem sequences; combinatorial designs; information theory; codes; AI algorithms; heuristic search

C. R. Categories: G.2.1, I.1.2, I.2.0

INTRODUCTION

A Steiner triple system (STS) of order v, STS(v), is a pair of sets (V, B) where $|V| = v$ and B consists of 3-subsets (triples or blocks) of V such that any 2-subset of V is included in exactly one block. A STS(v) exists iff $v \equiv 1$ or 3 (mod 6).

A STS(v) is cyclic if its automorphism group contains a v-cycle. A cyclic STS(v) exists for all $v \equiv 1$ or 3 (mod 6) except for $v = 9$. For more information about Steiner triple systems and other combinatorial designs the reader may consult Anderson (1990).

*Corresponding author.

While studying Steiner triple systems, Skolem (1957) asked whether it is possible to partition the set $\{1, 2, \ldots, 2n\}$ into n pairs (a_r, b_r) where $b_r - a_r = r$, for $r = 1, \ldots, n$. He showed that such partition is possible if and only if $n \equiv 0, 1 \pmod 4$. Later, such partitions were written as sequences, which are now known as Skolem sequences (SS). To illustrate, consider the case of $n = 4$, the sequence 1, 1, 4, 2, 3, 2, 4, 3 is equivalent to the partition of the set $\{1, 2, \ldots, 8\}$ into the pairs $(1, 2)$, $(4, 6)$, $(5, 8)$, $(3, 7)$.

Formally, a Skolem sequence of order n is an integer sequence $\mathrm{SS}(n) = (s_1, s_2, \ldots, s_{2n})$ of $2n$ integers satisfying the following conditions:

(1) $\forall k \in \{1, 2, \ldots, n\}$. $\exists$ exactly two elements s_i, s_j in $\mathrm{SS}(n)$ such that $s_i = s_j = k$.
(2) If $s_i = s_j = k$, $i < j$ then $j - i = k$.

Skolem (1958) also showed that the existence of a $\mathrm{SS}(n)$ implies the existence of a $\mathrm{STS}(6n + 1)$. By taking the pairs (a_i, b_i), $i = 1, \ldots, n$ produced from the sequence, we get the base blocks $\{0, i, b_i + n, 1, i = 1, \ldots, n\}$ for the $\mathrm{STS}(6n + 1)$. For example, the above sequence produces the base blocks for STS(25).

$\{\{0, 1, 6\}, \{0, 2, 10\}, \{0, 3, 12\}, \{0, 4, 11\}\} \pmod{25}$

Skolem sequences are special types of starters. A starter in the Abelian group Z_{2n+1} is a set $S = \{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_n, y_n)\}$ such that every non-zero element in Z_{2n+1} occurs as:

(a) an element of a pair in S, and
(b) a difference of a pair in S.

The above pairs $(1, 2)$, $(4, 6)$, $(5, 8)$, $(3, 7)$ form a starter in Z_9. Thus the existence of a Skolem sequence implies the existence of a starter, but the converse is not true. For more information on starters and their uses see Dinitz (1996). In fact, Skolem sequences and their generalizations are linked to several combinatorial designs, *e.g.*, Room squares and perfect one-factorization of complete graphs (Shalaby, 1991). It is also linked with several mathematical topics such as the Golden section and Wythoff game (Nowakowski, 1975).

The known applications of SS and their generalizations in the physical world include interference free missile guidance code (Eckler, 1960) and the construction of binary sequences with controllable complexity (Gorth, 1971). More details may be found in (Shalaby, 1996).

It seems, however, that there is no published algorithm for the construction of Skolem sequences. For small values of n (up to 8, say), SS may be generated by hand. For larger values of n exhaustive search may be used. However, practically speaking, exhaustive search is feasible for values up to 12. For larger values, no published algorithm is known. In this paper we use a hill-climbing algorithm to generate Skolem sequences so; (a) the designs that are related to SS can be generated and, as a special kind of starters (b) can be used by other researchers as an alternate method for generating the same objects starters generate, thus solving some of the still open problems in those areas. More specifically, the problem presented here is: given $n = 0, 1 \pmod 4$, generate at random a Skolem sequence using a hill-climbing algorithm. We exhibit several results of our findings.

HILL-CLIMBING

Generation of SS may be done by enumeration or exhaustive search. However, due to the very large search space for even small values of n, *e.g.*, for $n \geq 12$, unrestricted exhaustive search strategy is computationally very expensive. It is even unfeasible for larger values of n. A possible logical alternative is, therefore, to try a heuristic search method like hill-climbing.

Hill-climbing (H-C) algorithms are among the powerful heuristic search methods. They were used to search for optimal or near optimal solutions in some optimization problems like the traveling salesman problem. After the emerge of artificial intelligence (AI) applications, H-C algorithms accepted even a more wider range of interest. As compared to exhaustive search methods, H-C presents a 'smarter' method to tackle considerable sizable problems. Recently, there has been greater interest in applying H-C to combinatorial design problems as demonstrated in Gibbons (1996) and Tovey (1985). For example, Dinitz and Stinson (1987) used a hill-climbing algorithm to special kinds of starters that in turn produce the Room squares and one-factorizations. A trace of the successful applications of H-C in combinatorial design theory was reported in Gibbons (1993). In this paper we adopt H-C for the generation of SS. To do so we have to introduce the new concept of partial SS and to define its length.

PARTIAL SKOLEM SEQUENCES

A partial Skolem sequence $\mathrm{PSS}(n) = (s_1, s_2, \ldots, s_{2n})$ is a sequence of integers from $\{1, \ldots, n\}$ such that:

(1) Some integers in $\{1,\ldots,n\}$ occur in exactly two positions s_i, s_j,
(2) If $s_i = s_j = k$, $i < j$ then $j - i = k$.

PSS is in fact no more than a state during the SS generation process. In contrast to regular SS, it is possible to have voids in a PSS. For example the set 3, –, 2, 3, 2, –, –, – is a partial SS that may occur during the generation of a SS of order 4. Ultimately, the undefined elements will be: 4, 4, 1, and 1 (from left to right).

The length of a partial SS, denoted by L is the number of defined pairs in the sequence. In the previous example L will be 2. It is obvious that L for a completely defined SS is its order; n.

The generation of an initial partial SS is always possible. In the worst case, its length will be 1 only. This will happen if and only if the first two elements were consecutive ones, $x+1$ and x. As an example consider the generation of SS of order 4 and assume that the first two numbers were 4 and 3. In this case it is clear that conflict will occur and the generated partial SS will be of length 1 only.

THE ALGORITHM

The main idea is to try to build the sequence by selecting its pairs randomly from the set of integers $\{1,\ldots,n\}$. If conflict occurs, we retain the so far obtained 'good' elements and remove the 'bad' or 'noisy' element to continue the generation process. This cycle is repeated until the number of removals exceeds a preset threshold whence a new set of random numbers is tried. Thus the H-C meta algorithm may be stated as follows.

Let L(SS) be the number of pairs in a Skolem sequence of order n;

L(SS) := 0;
While (L(SS) $\neq n$) **do**
begin

Randomly arrange the set of integers $\langle 1,\ldots,n\rangle$

Generate an initial partial SS;

if L(SS) = n **then** stop **else**

Number-of-exchanges := 0;

While (Number-of-exchanges < threshold) **do**

```
        begin
            exchange the 'noisy' element;
            continue generation of the SS;
            if L(SS) = n then stop else
            Number-of-exchange := Number-of-exchange + 1;
        end;
        endwhile;
    end;
    endwhile.
```

The threshold parameter specifies how many exchanges we allow at any level before restarting the construction of a SS. Since the random number generator used is a recursive one, the new set obtained will never be identical to the previous one. This process is repeated until we get a valid SS. Since the necessary and sufficient condition for a SS to exist is that n must be $\equiv 0$, 1 (mod 4) as showed by Skolem (1957), we are sure from the very beginning about the existence of the SS. In this context we are lucky as compared to other cases where the existence of a particular combinatorial design is not guaranteed and whence another threshold is to be specified to completely abandon the search as in Gibbons (1993).

The performance of our algorithm depends to some extent on the chosen value as a threshold. Here we should notice that a too small value of threshold means the frequent generation of random numbers which are, by all means, computationally expensive. On the other hand, too large value of threshold means many exchanges which may result at last in a dead lock or a dead end. This may happen due to the fact that excessive exchange may result in a cyclic loop where after some iterations we return to the same state again and thus we may loop indefinitely. Our algorithm do prevent direct cycling when the exchange process results in exactly the previous state as explained later. Although it is possible to retain all the previously generated partial SS to avoid cycling, it is obvious that it is computationally extremely inefficient and in fact it violates the essence of H-C.

In view of the above discussion we experimented with some numerical values for a threshold as in Seah (1988) and Gibbons (1996).

Numerical experiments shows that a value of 6 is quite a reasonable threshold.

GENERATION OF THE SET OF RANDOM NUMBERS

It is well known that the generation of truly random numbers by a computer is not an easy task. In fact, the vast majority of the functions used for random number generation are pseudo ones. Running the same function using the same seed will result in obtaining the same set of random numbers. To overcome this difficulty we used the time when the program starts multiplied by a large integer number as the seed for the RAN function. Random numbers generated this way are then normalized to the order of the SS; n. When a duplicate number is obtained it is skipped and another number is tested. Although the last number in the list may be inserted, we prefer to obtain it as other numbers. A pseudo code for this procedure is as follows.

```
Procedure Generate-numbers (Num, n);
Begin
Seed := Current-time * 1234567;
Num[1, ..., n] := 0;
Row := 1;
Repeat until Row > n;
Begin
   x := RAN(Seed);
   While (x ≠ 0 and x ≠ 1) do
   y := Int(x*n + 1);
   If y ∉ Num[1 .. Row] then
      begin
      Num[Row] := y
      Row := Row + 1;
      end;
   endif;
   endwhile;
end;
end.
```

Comments:

- Seed is an integer variable used to call the RAN function.
- Num is an integer array of n elements to hold the normalized random numbers.

- n is the order of SS.
- Row is an integer variable used as an index for the Num array.
- RAN is the random number generator. It produces uniform random numbers $\langle 0, 1 \rangle$.
- Int is the truncation function.

GENERATION OF SKOLEM SEQUENCES

Here we generate the elements of the SS using the set of random numbers stored in Num array. We denote the element a_r in the pairs (a_r, b_r) as the element while the element b_r as the twin. The steps are as follows.

```
index := 1;
While (index ≤ n) do
begin
  Generate-numbers (Num, n);
  x := Num[index];
  SS [1] := x;
  SS [1 + x] := x;
  Number-of-exchanges := 0;
  While (Number-of-exchanges < threshold) do.
  begin
    index := index + 1;
    if index >  n then stop;
    x := Num[index];
    Check-SS (x, index1);
    if index1  ≠ 0 then
      SS [index1] := x;
      SS [index1 + x] := x;
    else
      Scan-array (x, remove, lastone);
      Num[index] := remove;
      index := index−1;
      Number-of-exchanges := Number-of-exchanges + 1;
endif;
```

```
    end;
    endwhile;
end;
endwhile;
```

Comments:

- index is an integer variable used as an index to the Num array.
- Check-SS is a procedure to check whether x can be stored in SS or not. At return of this procedure we may have index1 set to 0 to indicate that it is not possible to store the element in the SS array. If it is possible to store it, then index1 will hold the element number in SS.
- Scan-array is a procedure to scan the SS array and do the exchange process.
- remove is the removed element from SS due to exchange.
- lastone is the value of the last removed element.

EXCHANGE STRATEGY

An element may be stored in SS array only if its twin can be stored in the right place as well. If such a condition is not fulfilled we denote that element as being a 'noisy' element. For 'noisy' elements exchange must be done to be able to proceed with the SS generation. For that purpose, we scan the SS array for the *first* available empty element. We then check that the twin position is still within the boundaries of the SS array. If both conditions are met we do the exchange process by placing the 'noisy' element and its twin in the appropriate places. We remove the old element from the SS aray. If, however, it was not possible to store the twin within the boundaries of the SS array, we store the twin of the 'noisy' element in the *last* available element in the SS array and remove the element corresponding to the original element. The last exchanged element is always stored and if the exchange process will lead to remove the last exchanged element it is stopped to avoid cycling.

CYCLING AVOIDANCE

To avoid cycling *i.e.*, in step $i+1$ we do the same exchange we did in step i in a reverse direction, we use a flip-flop switch to scan the SS array from left to right in step i while scanning it from right to left in step $i + 1$. We also check that the element to be exchanged in not the last one exchanged in previous step. However, in very exceptional cases our scheme for cycling avoidance will fail, in this case we generate a new set of random numbers and start our algorithm again. To illustrate consider the following case when generating SS with $n = 8$. We reached the following partial SS and the last exchanged element was 2.

5		6	3	7	5	3	2	6	2	4	7	1	1	4	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

We are left with 8 only. In this case forward scanning leads to store 8 in second place and remove 2. On the other hand, backward scanning leads to the store of 8 in the last place and the removal of 2 as well. So in such very rare cases, we have to consider another set of random numbers as stated before.

EXAMPLES

Assume that an SS of order 5 is to be generated. Assume further that the set of random numbers generated was: 1, 3, 5, 4, 2. We show in a step by step fashion the progress of our algorithm as follows.

We can easily build the partial SS shown below.

1	1	3	5		3			5	
1	2	3	4	5	6	7	8	9	10

We are left with the two elements 4 and 2. Now we find it is impossible to store 4. Scanning forward will result in conflict with 5; while backward scanning will be in conflict with 3. Suppose we choose to remove the 5. Then the new partial SS will be:

1	1	3	4		3		4		
1	2	3	4	5	6	7	8	9	10

We are now left with 5 and 2. Now it is possible to store 5 and 2 and we obtain:

1	1	3	4	5	3	2	4	2	5
1	2	3	4	5	6	7	8	9	10

IMPLEMENTATION AND NUMERICAL EXPERIMENTS

The described algorithm was implemented in ANSI 78 FORTRAN known as FORTRAN 77 run on a Sun 690 server under UNIX OS. Table I shows the number of trials required to obtain SS of different orders. It is obvious that, in general, there is no trend observed between the number of trials and n.

In Appendix A examples of the generated SS of different orders are given.

ANALYSIS OF THE ALGORITHM

Obviously, hill-climbing is a local heuristic search (local improvement algorithm), thus it does not guarantee finding a global optimum or after running a certain number of times to find exhaustively all solutions. However, under the assumptions that:

(a) avoidance of cycling,
(b) all orderings of partial solutions are equally likely.

THEOREM (Tovey, 1985) *Under the assumption that all orderings are equally likely, the expected number of iterations of any local improvement algorithm is less than* $(3/2)e^n$.

Tovey also showed that, even though the local improvement algorithm is very fast to reach the local optima, it is unlikely to find the global optima. In our case, our algorithm satisfies (a) and (b) and, the number of global

TABLE I Generation of Skolem sequences

n	*# of trials*
4	1
8	1
12	1
16	18
20	4
24	22
28	35
32	227
36	31
40	319
44	527
48	137
52	1270
56	5420
60	634

optima *i.e.*, solutions of Skolem sequences increases exponentially with n. Abrham (1986) showed that the number of distinct Skolem sequences of order n, $\sigma_n \geq 2^{\lfloor n/3 \rfloor}$. These peaks seems sufficient for the practical purposes for finding design configurations when n is relatively small.

SKOLEM SEQUENCES AND NEW PERFECT ONE-FACTORIZATIONS OF K_{36}

A one-factorization of a complete graph K_{2n} is a partition of the edge-set of K_{2n} into $2n-1$ one-factors, each of which contains n edges that partition the vertex set of K_{2n}. A perfect one-factorization is a one-factorization in which every pair of distinct factors form a Hamiltonian cycle of the graph. We construct one-factorizations by using starters from Skolem sequences as we showed earlier. The following two new Perfect one-factorization for K_{36} were found by checking significantly less number of starters than that used by Seah and Stinson (1988):

15, 3, 9, 14, 3, 16, 12, 17, 4, 1, 1, 9, 4, 13, 2, 15, 2, 14, 12, 10, 8, 16, 11, 7, 17, 6, 13, 5, 8, 10, 7, 6, 5, 11 and 10, 14, 9, 13, 17, 8, 6, 16, 12, 5, 10, 9, 6, 8, 5, 14, 13, 15, 7, 11, 12, 17, 2, 16, 2, 7, 1, 1, 3, 4, 11, 3, 15, 4

CONCLUSIONS

Hill-climbing algorithms present attractive vehicle to the solution of many problems in different fields. Their versatility and ease of implementation are the key factors behind their wide spread and acceptance. Generating Skolem sequences of arbitrary order using hill-climbing is demonstrated in this paper. One of the critical parameters that greatly influence the performance of the hill-climbing algorithm is the value chosen as a threshold. The details of the algorithm are given. In addition to this, samples of the generated sequences are also given.

References

Abrham, J. (1986) Exponential Lower Bound for the Number of Skolem and External Langford Sequences. *Ars. Combinat.* (22).

Anderson, I. (1990) Combinatorial Design Construction Methods, Ellis Horwood Ltd.

Dinitz, J. (1996) Starters, CRC Handbook on Combinatorial Designs, Colbourn and Dinitz (Eds.).

Dinitz, J. H. and Stinson, D. R. (1987) A Hill-climbing Algorithm for the Construction of One-factorizations and Room Squares. *SIAM J. Algebraic Discrete Math.*, **8**, 430–438.
Eckler, A. R. (1960) Construction of Missile Guidance Codes Resistant to Random Interference. *Bell Sys. Tech.*, pp. 973–994.
Gibbons, P. (1996) Computational Methods in Design. CRC Handbook on Combinatorial Designs, Colbourn and Dinitz (Eds.).
Gibbons, P. and Mathon, R. (1993) The Use of Hill-climbing to Construct Orthogonal Steiner Triple Systems. *Journal of Combinatorial Designs*, **1**, 27–50.
Gorth, E. J. (1971) Generation of Binary Sequences with Controllable Complexity. *IEEE Trans. on Info. Theory IT-17*, **3**, 288–296.
Nowakowski, R. J. (1975) Generalizations of the Langford–Skolem Problem. M.Sc. Thesis, University of Calgary.
Seah, E. and Stinson, D. R. (1988) A Perfect One-factorization for K_{36}. *Discrete Math.*, **70**, 199–202.
Shalaby, N. (1991) Skolem Sequences: Generalizations and Applications. Ph.D. Thesis, McMaster University.
Shalaby, N. (1996) Skolem Sequences. CRC Handbook on Combinatorial Designs, Colbourn and Dinitz (Eds.).
Skolem, T. (1957) On Certain Distributions of Integers in Pairs with Given Differences. *Math. Scand.*, **5**, 57–68.
Skolem, T. (1958) Some Remarks on the Triple Systems of Steiner. *Math. Scand.*, **6**, 273–280.
Stinson, D. R. (1985) Hill-climbing Algorithms for the Construction of Combinatorial Designs. *Ann. Disc. Math.*, **26**, 321–334.
Tovey, C. A. (1985) Hill-climbing with Multiple Local Optima. *SIAM J. Alg. Disc. Math.*, **6**(3), 384–393.

APPENDIX A

Samples of the Generated Skolem Sequences

1) Regular Skolem sequence of order 20

15 9 11 17 7 4 19 5 6 4 9 7 5 11 6 15 20 10 8 14
17 18 16 12 13 19 8 10 3 1 1 3 2 14 2 12 20 13 16 18

2) Regular Skolem sequence of order 40

29 24 10 30 6 18 20 7 5 23 6 16 10 5 7 25 37 4 13 3
36 4 3 18 27 24 20 16 33 29 35 13 23 30 8 17 38 40 31 39
25 34 8 19 22 28 21 32 15 11 26 27 17 37 14 9 36 1 1 12
11 33 19 15 9 35 22 21 14 31 2 12 2 28 38 34 26 40 39 32

3) Regular Skolem sequence of order 52

18 36 27 38 16 20 52 42 48 17 1 1 33 23 30 39 49 6 18 3
16 34 3 6 37 20 17 5 29 27 8 28 5 31 14 45 23 36 8 46
2 38 2 40 30 33 44 47 14 42 43 51 15 50 39 34 48 29 52 28
41 37 35 32 31 49 26 15 19 12 21 4 10 25 22 4 24 7 11 9
45 12 10 40 7 46 13 19 9 11 44 21 26 43 47 32 22 35 25 13
24 41 51 50

4) Regular Skolem sequence of order 64

21 6 61 39 30 43 62 6 19 24 10 45 13 16 14 29 44 20 52 35
10 21 33 59 37 13 4 19 14 16 4 60 17 24 30 23 9 20 56 64
38 55 39 41 29 9 25 22 43 17 36 46 57 47 35 33 45 63 23 40
44 37 50 61 53 18 27 58 62 22 52 25 54 12 48 49 51 28 38 32
15 42 59 18 41 12 36 34 31 3 26 60 3 27 56 15 55 46 8 40
47 7 2 64 2 28 8 11 7 57 5 32 50 1 1 5 26 53 11 31
63 34 48 42 49 58 54 51

5) Regular Skolem sequence of order 84

69 39 3 53 11 3 57 16 78 63 40 26 30 42 22 11 44 35 13 75
70 41 74 16 68 18 67 80 20 32 55 13 65 38 71 45 22 26 62 14
39 60 30 18 83 77 24 10 20 76 40 73 35 14 19 42 53 10 31 43
44 32 41 57 72 58 82 64 61 69 24 38 63 19 54 46 23 21 48 66
45 84 52 81 51 55 78 59 79 31 70 47 68 67 75 37 74 65 21 23
62 60 43 36 33 71 50 80 34 8 56 29 49 17 27 5 28 8 1 1
5 46 77 58 73 76 48 83 54 61 17 64 37 25 52 51 72 33 47 36
29 27 34 9 28 66 59 15 82 6 4 12 9 7 4 6 50 2 25 2
7 49 15 12 81 84 56 79