Snake-In-The-Box Codes for Dimension 7

Krys J. Kochut

Department of Computer Science
University of Georgia
Athens, Georgia 30602-7404, USA

E-mail: kochut@cs.uga.edu

Abstract

In the n-dimensional hypercube, an n-snake is a simple path with no chords, while an n-coil is a simple cycle without chords. There has been much interest in determining the length of a maximum n-snake and a maximum n-coil. Only upper and lower bounds for these maximum lengths are known for arbitrary n. Computationally, the problem of finding maximum n-snakes and n-coils suffers from combinatorial explosion, in that the size of the solution space which must be searched grows very rapidly as n increases. Previously, the maximum lengths of n-snakes and n-coils have been established only for $n \leq 7$ and $n \leq 6$, respectively. In this paper, we report on a coil searching computer program which established that 48 is the maximum length of a coil in the hypercube of dimension 7.

1 Introduction

In this paper we present a new computational result for the "snake-in-the-box" problem. Specifically, we have developed a program which efficiently constructs all coils in hypercubes. Using this program, we have determined the maximum length of a coil in the hypercube of dimension 7. We used a substantially optimized version of a complete exhaustive search algorithm, similar to the one used in a different experiment, in which the maximum length of a snake in the hypercube of dimension 7 was determined [14].

Since the late 50's, hypercubes have been studied for their relevance to coding theory [11] and more recently due to the construction of parallel computing systems with hypercube communication topologies. Harary *et al.* [10] presented a survey of known theoretical results concerning hypercubes. Coils (simple cycles without chords) in hypercubes, as opposed to snakes (simple paths without chords), have received the most attention in the literature [7, 1, 10, 19, 17]. Both coils and snakes have various applications, such as error-detection in analog-to-digital conversion [13]. Generally, the longer the snake, the more accurate the conversion will be. Additionally, snakes are related to algorithms used for disjunctive normal form simplification and for electronic combination locking schemes; again, the longer the snake, the more useful it is in the application [13].

2 Background and Definitions

We will consider only simple graphs $G(V, E)$, i.e. graphs with neither multiple edges nor loops. A *complete graph* on n nodes, denoted K_n, is a graph in which every two distinct nodes are adjacent (connected by an edge).

A *path* in G is a sequence of nodes $v_0, v_1, ...v_n$, in which v_{i-1} and v_i $(1 \leq i \leq n)$ are adjacent (in G). Such a path is said to have endpoints v_0 and v_n, and to have length n. A *simple path* is a path with no repeated nodes.

A *chordless path* in G is a simple path $v_0, v_1, ...v_n$ in G, such that v_i and v_j are never adjacent when i and j differ by more than 1.

A *cycle* in G is a sequence of nodes $v_0, v_1, ...v_n, v_0$, where $v_0, v_1, ...v_n$ is a path and v_n and v_0 are adjacent. Such a cycle is said to have length $n + 1$. A *simple cycle* is a cycle $v_0, v_1, ...v_n, v_0$, such that $v_i \neq v_j$ if $i \neq j$.

A *chordless cycle* in G is a simple cycle $v_0, v_1, ...v_n, v_0$, in which v_i and v_j are never adjacent in G when i and j differ by more than 1 (mod $n + 1$).

The *cartesian product* of two graphs G_1 and G_2, denoted $G = G_1 \times G_2$, is a graph G, where the set of nodes of $G = V_1 \times V_2$ and two nodes (u_1, u_2) and (v_1, v_2) of G are adjacent in G if and only if either $u_1 = v_1$ and u_2 and v_2 are adjacent in G_2, or $u_2 = v_2$ and u_1 and v_1 are adjacent in G_1.

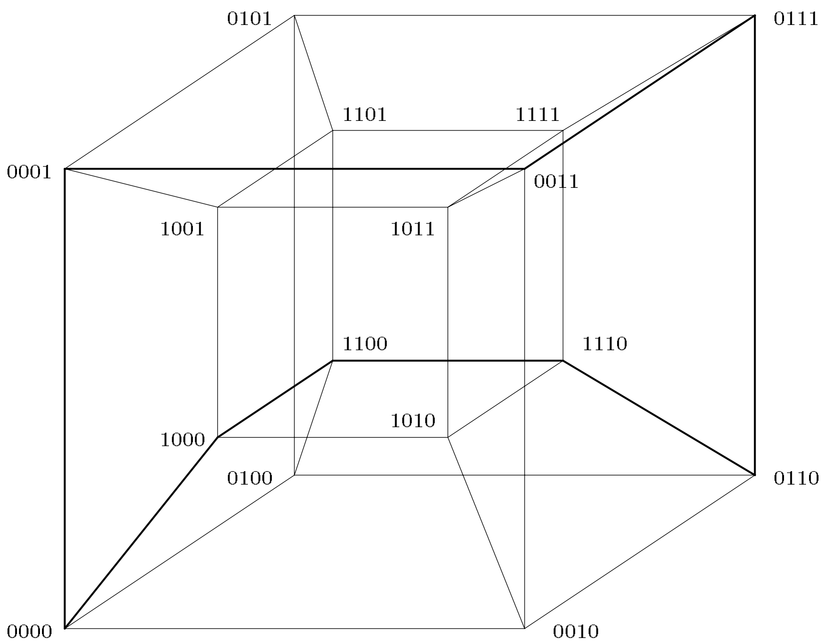

Figure 1: A coil in Q_4

The $n-dimensional\ hypercube$, denoted Q_n, can be defined inductively as in [10] by:

$$
\begin{aligned}
Q_1 &= K_2, \\
Q_n &= K_2 \times Q_{n-1}.
\end{aligned}
$$

The nodes of Q_n can be represented as 2^n vectors of binary digits, each of length n. Two nodes are adjacent in Q_n if they differ in exactly one coordinate. For example, in Q_4, the node 1010 is adjacent to 1000, 1011, 1110, and 0010.

An *n-snake* is a chordless path in Q_n. An *n-coil* is a chordless cycle in Q_n. The *length* of an n-snake (n-coil) is the number of edges in the n-snake (n-coil), i.e., just its length as a path (cycle).

A Gray code is a Hamiltonian path or cycle in the hypercube. Using the binary vector representation, one Gray code for Q_4 is:

```
0000   0001   0011   0010   0110   0111   0101   0100
1100   1101   1111   1110   1010   1011   1001   1000
```

For dimension $n \geq 3$, every Gray code contains at least one chord. For instance, in the Gray code above, 1010 and 1000 are adjacent in Q_4 but not consecutive in the code, so this edge is a chord. Figure 2 shows a maximum length coil in Q_4 containing 8 edges. The nodes of the coil are:

```
0000   0001   0011   0111   0110   1110   1100   1000   0000
```

Great interest has developed in determining maximum lengths of coils and snakes in Q_n (see for example [11, 6, 16, 5, 12, 8, 4, 7, 1, 19, 2, 17]). In many of these papers, the term *snake* has been used to mean an n-coil. In the present paper, we follow the notation introduced in [10] and use the terms *coil* and *snake* to refer to an n-coil and an n-snake, respectively. We use c_n and s_n to denote the length of a maximum n-coil and of a maximum n-snake, respectively.

Finding long coils and snakes is such a computationally difficult problem that the values of c_n and s_n have previously been established only for n up to 6 and 7, respectively. The result for s_7 was reported in [14]. The result for c_7 is Theorem 1, below. The known maximum lengths of n-coils and n-snakes are summarized in Table 1.

n	s_n	c_n
1	1	2
2	2	4
3	4	6
4	7	8
5	13	14
6	26	26
7	50	48

Table 1: Maximum Lengths of Snakes and Coils

To date, only upper and lower bounds have been reported in the literature for c_n, where $n \geq 7$, and s_n, where $n \geq 8$. Table 2 shows the lower bounds for coils and snakes for $8 \leq n \leq 11$. The result for s_8 was obtained computationally by using the genetic algorithm and reported by Potter *et al.* [15]. Other results are by Abbott and Katchalski [3], and Even [9].

n	$l.b.$ for s_n	$l.b.$ for c_n
8	89	88
9	168	170
10	322	324
11	618	620

Table 2: Known Lower Bounds on Maximum Lengths of Snakes and Coils

Clearly, $c_n - 2 \leq s_n$, since one can always form an open snake by deleting a node from a coil. The lower bounds for s_n for n = 9, 10, and 11 follow from this. Additionally, a lower bound for c_7 of 48 was computationally established by W.L. Eastman, as reported in [9].

A number of authors have derived upper bounds for c_n. Solov'jeva [18] reported bounds for $n \geq 7$, which were recently improved by Snevily [17] for $n \geq 12$. They give bounds for all $n \geq 7$ ($n \geq 12$, in [17]) which are $< 2^{n-1}$ but asymptotic to 2^{n-1}. The upper bounds do not seem to be as good as the lower bounds, based on the data we now have for $n \leq 7$.

3 Coil Searching Algorithm

The enumerative algorithm works by selecting new nodes for exploration in Q_n in a depth-first search. The algorithm attempts to extend a path, so that it is an n-snake, one node at a time. On each extension attempt, a check is performed to find out if the current path may be closed to form an n-coil. Since in our experiment we considered Q_7, our explanations will be based on hypercube nodes represented as binary vectors of length 7.

All enumerated snakes start at the origin, 0000000, and are extended at the other end. If a coil may be closed by adding two additional edges (leading back to the origin, i.e. 0000000), the length of the coil is recorded. When a snake cannot be extended it is considered a dead-end (such a dead-end snake is a maximal snake). A dead-end snake is discarded and other snakes are considered. Backtracking occurs at each dead-end snake, so that a whole tree of possibilities is explored.

We utilized the following symmetry optimization, similar to that employed in the earlier experiment which established the maximum length of a snake in Q_7 [14]. There are $n!$ symmetries of the n-cube which leave the origin fixed, corresponding to the symmetric group of all permutations of the n coordinates. Each maximal path makes use of all n coordinates in its transitions, and therefore cannot be left fixed by any of these symmetries of the n-cube except for the identity. That is, each path belongs to a class of $n!$ paths which are equivalent, and in particular are all of the same length. In order to explore exactly one path in each equivalence class, the search algorithm considers only paths which are *canonical* in the sense that the first occurrence of a 1 in each position follows the linear order of least significant to most significant. Following this rule for $n = 7$, the next node after 0000000 must be 0000001, rather than 1000000 or any of the other five nodes adjacent to the origin. Later on, say when 1's have appeared in each of the first four positions but not in any of the last three, the next position to contain a 1 for the first time must be the fifth. The current path is extended by moving to an adjacent node that is not adjacent to any other nodes in the current path. When dead-ends are encountered, the algorithm backtracks.

The coil searching algorithm described here includes an additional optimization for pruning the search tree. The optimization eliminates paths which can never possibly be closed to form coils (by returning back to the origin). Since the coil searching algorithm starts enumerating canonical paths at node 0000000, there are 6 nodes that can eventually lead back to the origin (one of the 7 neighboring nodes, specifically 0000001, is used to initially extend the path from the origin, and therefore is excluded as a potential return node). Each time a path is extended by one node, all of the neighbors are marked and if any of them are also neighbors of the origin, the number of potential return nodes is decremented accordingly. Obviously, if the number of potential return nodes drops to zero, the current path may be discarded, even though it may be extended further.

The pseudo-code of the main algorithm is shown in Fig. 3. The algorithm uses two stacks, one for the nodes visited "so far" (representing the current path) and one for the pivot values, indicating the highest dimension already explored. (This value is necessary for implementing the symmetry optimization, as described above.) Procedure `mark_neighbors` marks the neighbors of the current node and decrements the number of potential return nodes, whenever necessary.

```
procedure coilsearch( depth: integer );

  var next, curr: node;
      i, pivot:   integer;

begin

  curr_node := stack_top(nodestack); pivot := stack_top(pivotstack);

  mark_neighbors( curr_node );
  remember nodes marked at this level (later called just marked);

  if  there are no neighbors which were just marked  or
      number of possible return paths is zero  then
    return
  else begin
    for all i from 0 to pivot do
      if  the i-th neighbor was just marked then
      begin
        if can_close_coil then save_coil(depth);
        push(the i-th neighbor, nodestack); push(pivot, pivotstack);
        search( depth+1 );
        pop( nodestack );  pop( pivotstack );
         if  empty( nodestack )  then  return
      end;

    if  pivot < dim-1  then
    begin
      pivot := pivot + 1;
      if  pivot-th neighbor was just marked  then
      begin
        if can_close_coil then save_coil(depth);
        push(the pivot-th neighbor, nodestack);
        push(pivot, pivotstack);
        search( depth+1 );
        pop( nodestack );  pop( pivotstack );
        if  empty( nodestack )  then  return
      end
    end
  end

  unmark nodes marked at this level;

end;
```

Figure 2: Pseudocode of Our Coil Search Algorithm

4 Results of the Coil Search

Since we anticipated a very long running time of the exhaustive coils search, the program had to "checkpoint" itself (save its state) frequently. Using the recursive search algorithm shown in Fig. 3, it would be difficult to to restart the program from any given checkpointed state. Thus, the recursive algorithm was converted to an equivalent, non-recursive, stack-based version. The converted algorithm was coded in the C programming language, and then optimized for speed. The non-recursive version implemented the same basic algorithm, but instead of using the system stack, as is implicitly done in recursive procedures, we elected to maintain a number of stacks ourselves.

The enumerative coil search algorithm was run to look for canonical coils in Q_7. The starting node was fixed at 0. To speed up the coil searching process, the overall search tree was expanded to the depth of 7. At that level, the search tree contained 12 unexplored branches. Since each of the search-trees rooted at the 12 unexplored nodes at depth 7 was not connected to any other search-tree, the searches of each branch were completely independent. Because of this, they could be explored in parallel on independent computers. The computation was carried out on a network consisting of 5 SUN Microsystems SparcCenter 1000's (with two processors each). The computation lasted a bit more than one month.

Theorem 1 *From exhaustive search,* $c_7 = 48$.

Proof: Since our algorithm performed an exhaustive search, the complete computation enables us to conclude that the longest coil in Q_7 has length 48. It is only necessary to know this for canonical coils, since any coil is equivalent to one of the canonical coils (and equivalence preserves coil length). Thus, we have established that $c_7 = 48$. The program found 67488 canonical coils of length 48. A sample coil of length 48 from each of the 12 independent computations is presented in Table 3. □

5 Conclusions

Our coil searching program has established the value of c_7 to be 48. This is in agreement with our earlier experiment [14], which established the value of s_7 and improved the upper bound on c_7 by lowering it to 50. (It was already known that 48 was a lower bound for c_7.)

References

[1] H.L. Abbott and M. Katchalski. On the Snake-in-the-Box Problem, *J. Combin. Theory* **45** (1988), 13–24.

[2] H.L. Abbott and M. Katchalski. Further Results on Snakes in Powers of Complete Graphs, *Discrete Math.* **91** (1991), 111–120.

[3] H.L. Abbott and M. Katchalski. On The Construction Of Snake In The Box Codes, *Utilitas Math.* **40** (1991), 97–116.

[4] L.E. Adelson, R. Alter, and T.B. Curtz. Long snakes and a characterization of maximal snakes on the d-cube, *Congr. Numer.* **8** (1973), 111–124.

[5] L. Danzer and V. Klee. Lengths of Snakes in Boxes, *J. Combin. Theory* **2** (1967), 258–265.

[6] D.W. Davies. Longest -Separated- Paths and Loops in an N Cube, *IEEE Trans. Electronic Computers* **14** (1965), 261.

[7] K. Deimer. A New Upper Bound for the Length of Snakes, *Combinatorica* **5** (1985), 109–120.

[8] R.J. Douglas. Upper Bounds on the lengths of circuits of even spread in the d-cube, *J. Combin. Theory* **7** (1969), 206–214.

[9] S. Even. Snake in the Box Codes, *IRE Transactions on Electronic Computers* **EC–12** (1963), 18.

[10] F. Harary, J. P. Hayes, and H. J. Wu. A survey of the theory of hypercube graphs, *Comput. Math. Applic.* **15** (1988) 277–289.

[11] W.H. Kautz. Unit-Distance Error-Checking Codes, *IRE Trans. Electronic Computers* **7** (1958), 179–180.

[12] V. Klee. A method for constructing circuit codes, *J. Assoc. Comput. Mach.* **14** (1967), 520–538.

[13] V. Klee. What is the maximum length of a d-dimensional snake?, *Amer. Math. Monthly* **77** (1970), 63–65.

[14] K.J. Kochut, J.A. Miller, W.D. Potter, and R.W. Robinson. Hunting For Snake-In-The-Box-Codes, *Annals of Mathematics and Artificial Intelligence* (1994) (under review).

[15] W.D. Potter, R.W. Robinson, J.A. Miller, and K.J. Kochut. Using the Genetic Algorithm to Find Snake-In-The-Box Codes, *7-th Intl. Conf. on Industrial & Eng. Applic. of Artificial Intelligence and Expert Systems*, Austin, Texas, (1994), 421–426.

[16] R.C. Singleton. Generalized Snake-in-the-Box Codes, *IEEE Trans. Electronic Computers* **15** (1966), 596–602.

[17] H.S. Snevily. The Snake-in-the-Box Problem: A New Upper Bound, *Discrete Math.*, (1993) (to appear).

[18] F.I. Solov'jeva. An Upper Bound for the Length of a Cycle in an n-Dimensional Unit Cube, *Discret. Analiz.* **45** (1987).

[19] J. Wojciechowski. A new lower bound for Snake-in-the-Box codes, *Combinatorica* **9** (1989), 91–99.

No.	Sequence
1	0 1 3 7 15 13 12 28 30 26 27 25 57 56 40 104 72 73 75 107 111 110 46 38 36 52 116 124 125 93 95 87 119 55 51 50 114 98 66 70 68 69 101 97 113 81 80 16 0
2	0 1 3 7 15 13 29 28 30 26 27 59 57 41 40 42 46 38 36 37 53 55 119 87 86 84 68 76 78 74 75 107 111 109 125 124 120 88 89 81 113 97 96 98 114 50 48 16 0
3	0 1 3 7 15 14 10 26 27 25 29 28 20 22 54 50 51 49 53 117 125 121 120 88 80 81 83 87 95 94 126 110 108 44 45 41 43 107 75 73 77 69 68 70 66 98 96 32 0
4	0 1 3 7 15 14 12 28 29 25 27 26 18 22 54 55 51 115 83 87 95 94 126 122 120 121 125 117 116 84 68 69 77 73 72 74 66 98 102 103 111 107 43 41 45 37 36 32 0
5	0 1 3 7 15 14 30 28 29 25 27 59 58 56 40 41 45 37 53 52 54 38 34 98 96 97 113 121 125 124 126 110 111 103 119 87 86 70 68 69 77 73 75 74 90 88 80 16 0
6	0 1 3 7 15 31 27 26 24 28 20 21 53 49 51 50 54 62 46 42 43 41 105 104 120 122 123 127 111 103 99 98 66 74 78 94 86 87 83 81 89 93 77 69 68 100 36 32 0
7	0 1 3 7 15 31 29 28 24 26 18 22 54 55 53 49 57 59 43 42 40 44 36 100 101 97 99 98 114 122 120 124 125 127 111 110 78 76 77 73 75 91 83 87 85 84 80 64 0
8	0 1 3 7 15 31 30 28 24 25 57 59 51 50 54 52 53 37 45 44 46 42 10 74 90 122 126 124 125 93 77 73 105 107 111 103 102 98 96 112 113 81 83 87 86 84 68 4 0
9	0 1 3 7 15 31 63 62 60 56 48 49 113 121 123 91 75 73 77 69 85 21 20 22 18 26 10 42 43 41 45 37 36 100 116 118 114 98 99 103 111 110 78 94 92 88 80 64 0
10	0 1 3 7 6 14 10 26 27 25 29 28 20 52 53 55 51 50 34 98 102 103 101 97 113 81 83 82 86 94 95 127 125 124 120 104 40 44 45 47 43 107 75 73 77 76 68 64 0
11	0 1 3 7 6 14 12 13 29 31 27 26 18 50 54 55 53 49 57 121 105 73 75 74 66 98 99 115 83 87 85 69 68 100 116 112 80 88 92 94 126 127 111 47 43 42 40 32 0
12	0 1 3 7 6 14 30 31 29 25 24 56 58 59 43 41 45 37 101 100 68 84 20 52 54 55 119 87 83 91 90 74 72 73 77 79 111 110 126 124 125 121 113 112 114 98 34 32 0

Table 3: A Sample of Coils in Q_7 of Length 48