

Discovering snake-in-the-box codes through pruning based on the exemplary solutions of an Evolutionary Algorithm

Daniel R. Tuohy, Walter D. Potter and Darren A. Casella

Abstract—We present a method for searching for simple achordal open paths (snakes) in n -dimensional hypercube graphs (the box). Our technique first obtains exemplary snakes using an evolutionary algorithm previously responsible for defining the best-known lower bounds in n -cubes for $n = 9, 10, 11,$ and 12 . These snakes are used to define a pruning model that constrains the search space. A depth-first search search of the constrained solution space has established a new lower bound for the length of the longest snakes in the 9 and 10 dimensional hypercubes.

I. INTRODUCTION

The snake-in-the-box problem is that of establishing the longest *induced path* in an n -dimensional hypercube. The longest such path for the dimension-4 hypercube is illustrated in Figure 1. The problem was first described in a paper by Kautz in the late 50's, and was noted for its relevance to coding theory [7]. More recently, snakes have been of use in algorithms for disjunctive normal form simplification, electronic combination locking mechanisms, error-detection, and analog-to-digital conversion [8][9].

Many computational search techniques have been employed to discover lengthy snakes and coils. These include exhaustive search [4][9], Genetic Algorithms [10], distributed computing [6], and other evolutionary algorithms [3]. Our technique for finding lengthy snakes is a constrained depth first search. The depth first search is constrained considerably using the methods described by Kochut [9]. However, this still leaves us with a search space that is intractable by exhaustive search. Consequently, we have introduced novel pruning measures to further constrain the search space. We first obtain very good solutions using the evolutionary algorithm for the snake-in-the-box problem described in [3]. These snakes are then used to define criteria to which snakes must adhere in every point of the search. These criteria are defined in section IV-B.

II. BACKGROUND AND TERMINOLOGY

We use Q_n to denote the n -dimensional hypercube, which is defined inductively as the Cartesian product of Q_1 and Q_{n-1} . It is useful to think of Q_1 as a line (two constituent nodes), Q_2 as a square (four), Q_3 as a cube (eight), and Q_4 as the sixteen-node graph in Figure 1. For $n > 4$, meaningful visualisation is more tricky.

There are 2^n nodes in Q_n that can be represented as vectors of binary digits. The nodes are labeled in such a

D.R. Tuohy, W.D. Potter and D.A. Casella are with the Artificial Intelligence Center, University of Georgia, Athens, GA 30602-7415, USA (phone: 404-314-8467; email: danielr2e@gmail.com).

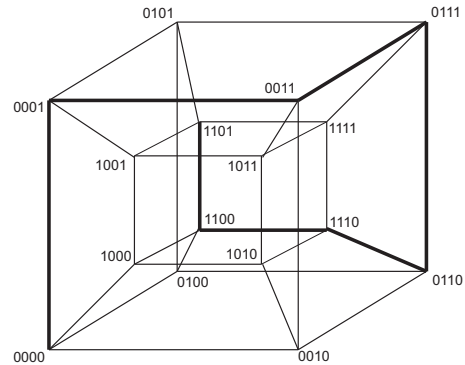


Fig. 1. A maximal length snake in Q_4

way that the binary vectors of adjacent nodes always differ by exactly one bit, as in Figure 1.

An *induced path* in Q_n , as defined in [5], is a sequence of nodes P such that for any $u, v \in P$, if u and v are adjacent in Q_n then they are also adjacent in P .

A path is expressed in *node sequence* representation if it is a vector of base-10 integers corresponding to the binary labels on each node in the path. The node sequence of the snake in Figure 1 is $\{0\ 1\ 3\ 7\ 6\ 14\ 12\ 13\}$.

A path is expressed in *transition sequence* representation if it is a vector of integers in the range $0..n-1$ [1]. These integers correspond to the index of the bit in the bit string that was flipped to grow the snake from one node to the next. The index of the least significant bit is 0 and that of the most significant bit is $n-1$. The transition sequence of the snake in Figure 1 is $\{0\ 1\ 2\ 0\ 3\ 1\ 0\}$.

The term “Snake in the Box” (derived from the visualization of a chain as a unit-radius tube) originally designated induced cycles, or closed paths (also called closed chains), in a graph [7]. We adopt the terminology introduced by Harary in [5], who assigns the term “coil” to simple cycles and the term “ n -coil” to coils of maximal length in Q_n . The terms “snake” and “ n -snake” are used in exactly the same way to designate induced open paths.

The lengths of n -snakes and n -coils up to $n = 6$ were computed via exhaustive search by Davies [4]. Those for $n = 7$ were determined by the exhaustive algorithms of Kochut [9] and the Genetic Algorithm of Potter [10], which we will be building upon. Table 1 shows the lengths of the n -snakes and n -coils for $n \leq 7$, as well as current best-known lengths of longest snakes and coils in dimensions 8 through 12 [2][3][4][5][9][10].

Dimension	n -snake	n -coil
Q ₁	1	2
Q ₂	2	4
Q ₃	4	6
Q ₄	7	8
Q ₅	13	14
Q ₆	26	26
Q ₇	50	48
Q ₈	97	96
Q ₉	186	180
Q ₁₀	358	344
Q ₁₁	680	630
Q ₁₂	1260	1238

TABLE I

LONGEST SNAKES AND COILS IN Q_n . FOR $N \geq 8$, SOLUTIONS ARE ONLY THE CURRENT BEST-KNOWN.

III. PBSHC: AN EVOLUTIONARY ALGORITHM FOR OBTAINING MODEL SNAKES

A Population-Based Stochastic Hill-Climber (PBSHC) was used to generate our model snakes. This algorithm is described in more detail in [3]. The PBSHC evolves a population of snakes from a zero or small initial length to some maximum length. Each individual in the population consists of a sequence of integers that represents the node sequence of a snake, or valid path through the hypercube, in the dimension being searched. These individuals are initialized as either a snake of length zero, that is consisting of only the zero node, or seeded with a pre-existing snake of choice. Following initialization, the evolutionary cycle begins its first generation. Each generation begins with a fitness evaluation. The fitness function used is based on both the length and the ‘tightness’ of the snake. The tightness of a snake, as defined for the PBSHC, is a measure of how many nodes are left available in the hypercube after subtracting all those nodes that are disqualified either by already being in the snake or by being adjacent to another node in the snake. The choice of tightness as a component of the fitness function was inspired by the idea that tighter snakes, since they make more efficient use of nodes, might have more room to grow.

Initial attempts of integrating both length and tightness into a fitness function met with limited success. Many combinations of linear and non-linear factors of length and tightness for the fitness function were tried. While they each performed well in some periods of evolution, they would inevitably perform worse in others. Once the average fitness of the population slowed, the diversity of the population would fall and the fitness function would converge to a local optimum. Our first attempt at solving this problem was to develop an adaptive fitness function that would balance between the importance of tightness and length, using the diversity of the population as the balancing factor. This technique also failed, reacting to changes in diversity too quickly in some cases and not quickly enough in others. Our second attempt was a more simplistic compromise between the two objectives. Length was set as the overriding objective within each generation,

and tightness delegated to an ordered ranking factor of the rank-based selection within each generation. Since all snakes that were able to grow in the previous generation have the same length in the current generation, tightness becomes the only distinguishing factor of the fitness function.

After the individual fitness of each of the snakes in the population is determined, the population is subjected to rank-based selection based upon each snake’s fitness. In order to keep the population constant throughout the entire run, the remaining places available in the population of the next generation are filled starting again with the most fit individuals. For example, if the selection percentage is 90%, the individuals are ranked by fitness and the first 90% are selected. This still leaves 10% of the next generation empty, so the first 10% from the current generation are used to finish the selection operation.

After selection, the growth operator grows each snake in the population by one step each generation, connecting each snake’s end node to one of its adjacent nodes that has not been disqualified by already being in the snake, or by being adjacent to some node that is in the snake. While bi-directional growth was used during preliminary trials in dimension eight, unidirectional growth was chosen for this implementation to best allocate computational resources. This operator can be seen to perform a stochastic hill-climbing process on each snake in the population as the choice of which adjacent node to connect to is based on random selection from the available nodes. A choice was made early to grow all snakes in the population instead of only the snake of best fitness (note: enhancing the best individual in a population is a common approach in hybrid genetic algorithms). This choice was also based on results of a comparison of these two approaches in an earlier GA implementation. Growing all the individuals within the population also works well in conjunction with the fitness function which requires that all snakes in the population of a given generation be of the same length in order to function properly. The fitness function consists of the sum of the snake’s length and normalized tightness. This results in the fitness function simplifying to a function of tightness alone as the length component of the fitness value will dominate for snakes of different lengths, yet cancel for snakes of the same length. This also results in the automatic elimination of snakes that can no longer grow, allowing their place in the population to be reallocated to other snakes that are still capable of growth.

The choice of parameter settings was found to be of key importance to the performance of the PBSHC and these settings were tuned extensively in dimension eight before modified to run in dimensions nine through twelve. Our previous experience with genetic algorithm snake hunters supported the application of lower-dimension parameter settings as a baseline for higher-dimension runs. The fitness function is set to the sum of length and normalized tightness. Normalized tightness was defined as the number of nodes remaining available divided by the total number of nodes

in the n -dimensional hypercube. Rank-based selection was chosen in order to maximize diversity within the population. While both unidirectional and bi-directional growth were used in the dimension-eight trials, the relatively memory-conservative, unidirectional implementation was chosen for the higher-dimensional runs in order to maximize potential population sizes for dimensions nine through twelve. Because this particular implementation’s chromosome size is constant, the use of unidirectional growth instead of bi-directional growth allowed the required memory for each population size to be reduced by half. Population sizes from one hundred through ten thousand were run in trials using mutation. Larger populations were prohibitively time-consuming using mutation, especially for dimensions eleven and twelve.

The best results were achieved using populations of ten thousand, a selection percentage of ninety percent, and by seeding each population with highly-fit, lower-dimension snakes at startup. Using a technique where the best snakes found in each dimension were used as seeds for the next higher dimension, the PBSHC was able to find new lower bounds for snakes in dimensions nine through twelve and new lower bounds for coils in dimensions nine through eleven. In order to generate good seeds for dimension nine, a bootstrap solution was used. This method involved cutting a dimension eight, length-97 snake back to its length-50 root, corresponding to the longest snake in dimension seven, and running an exhaustive search on that snake to generate a pool of seventeen distinct length-97 snakes. These snakes were then used to seed the dimension-nine runs. Subsequently, the best snakes found in dimension n were used as seeds for dimension $n+1$. For the purposes of the method described in this paper, we have used PBSHC to generate populations of very long snakes.

IV. CONSTRAINING THE SOLUTION SPACE FOR A DEPTH-FIRST SEARCH

To explore the solution space, we attempt to grow a snake until it reaches a dead end. The search then backtracks until further progress can be made in a different direction. This process is repeated until every snake has been explored. Unfortunately, there are far too many possible snakes to try every one, and we are forced to constrain the solution space artificially. We will describe five methods by which we constrain the solution space so that it may be fully explored by a depth first search. The first two, described in the following section, were first implemented by Kochut for exploring Q_7 .

A. Kochut’s Constraints

The constraints implemented by Kochut are independent of model snakes and were first used in an exhaustive snake searching algorithm devised for the dimension 7 hypercube. Neither of these restrictions can possibly prevent the discovery of a maximal length snake, yet they drastically increase search efficiency.

Always begin at node 0. To understand why this does not prevent us from finding possible maximal length snakes, it is helpful to consider a snake in transition sequence (see Sect. II). From a snake starting at node 0 in Q_n , one can obtain n^2 identical snakes in node sequence simply by applying the corresponding transition sequence beginning at every node in Q_n . In dimension 9, by only searching for snakes originating at node 0 we eliminate 99.61% (255/256) percent of the solution space from consideration.

Only consider snakes in canonical form. Snakes in canonical form are those which only use higher-order dimensions after every lower-order dimension has been used at least once. Hypercubes have $n!$ symmetries, and canonical form is but one of these. Again, consider a snake in transition sequence. By swapping, say, all of the 8’s and 1’s, we obtain an identical snake with a different node sequence. It simply uses the dimensions in a different order. In dimension 9, by only searching for snakes in canonical form we eliminate 99.972% ($(n!-1/n!)$) of the solution space from consideration.

B. Constraints Based on Model Snakes

The solution space afforded by Kochut’s approach is still far too massive to be searched exhaustively in dimensions greater than 7. We therefore introduce further constraints that no longer guarantee the maximal length snake will be found. Consequently, we can only find new lower bounds for maximal snake lengths, rather than absolute bounds. The following three constraints ensure that every snake considered in the depth first search adheres to a set of parameter boundaries defined by the model snakes found via PBSHC. In essence, we are making a much more strict definition of a “dead end” in the search. Backtracking now occurs not only when there are no available adjacent nodes, but also when all available nodes cause the snake to escape the parameter boundaries.

Tightness. Tightness is the degree to which different nodes in the snake render identical nodes inaccessible and is proportional to the number of pairs of included nodes with a hamming distance of 2. For example, nodes 00000000 and 000000011 both render node 000000001 inaccessible because they differ at exactly 2 bits. The “tighter” a snake is wound, the more nodes remain available at each step in the snake. We express tightness as “the number of unavailable nodes per node in the snake”. If there are 18 unavailable nodes at node 4 in the snake, the tightness of the snake is 4.5. It tends to be desirable to keep this number low. However, we have found that our model snakes do not always choose the tightest possible path at every step. We therefore prune branches both when snakes are too loose and when they are too tight. Figure 2 is an example of tightness boundaries that have been defined by model snakes in Dimension 9.

Tightness Rate-of-change. Consider a snake that is on the tighter edge of the tightness boundary defined by our model snakes at the k^{th} step in the snake. For the next several steps, the search has license to take inadvisable turns that needlessly eat up available nodes in the hypercube since it will still stay inside the tightness boundary. To eliminate this possibility,

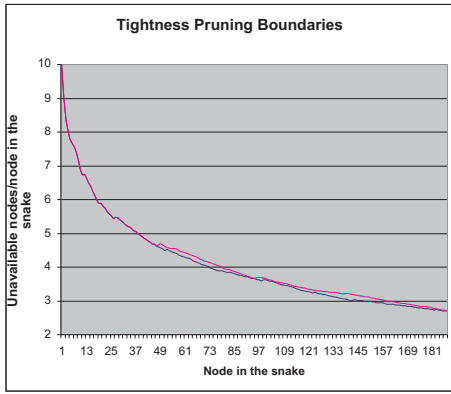


Fig. 2. The tightness boundaries defined by the model snakes found via PBSHC. Snakes are pruned as soon as they escape these boundaries.

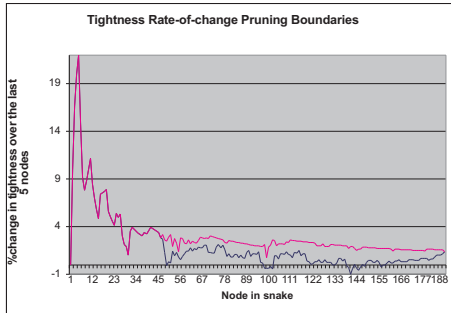


Fig. 3. The tightness rate-of-change boundaries defined by the model snakes found via PBSHC. Snakes are pruned as soon as they escape these boundaries.

we only allow a snake’s tightness to increase or decrease at a certain rate. We do this by extracting the “percent change in tightness over the last 5 steps” at each step in our model snakes. This corresponds to the rate of change in tightness at each step, and we do not allow snakes to tighten or loosen faster than any of the model snakes. Figure 3 shows the tightness rate-of-change boundaries that have been define by model snakes.

Maximal Dimension Cumulative Usage. This term refers to the number of times each dimension has been “used” in the snake. A dimension is “used” if it has been flipped from a 1 to a 0 or vice-versa in the course of creating the path. Usage is equivalent to the number of occurrences in the transition sequence. We have observed a great deal of homogeneity in the Dimension Cumulative Usage of model snakes. This is in part due to the fact that all model snakes are in canonical form, which ensures that the dimensions are ordered the same way.

In the Q_9 model snakes, the highest dimension is always used only once. Additionally, dimension 6 is always used more than dimension 5 and dimension 4 is always the most used dimension. Each dimension has a range of about 10 uses from which it never deviates in the model snakes. Currently, we only prune snakes which overuse any of the dimensions and do not prune for underuse.

V. RESULTS

We have two methods of control over how strictly we constrain the solution space. The first is choosing a seed snake. Past research has found it useful to choose a portion of a best known snake from a lower dimension to seed a search algorithm [3]. A seed, in effect, sets in stone the initial section of the path, reducing the search space.

The second is the number of snakes used to define the pruning model. Fewer snakes intuitively results in narrower pruning boundaries because there is less variability among fewer snakes.

A. Dimension 9

A base of 344 snakes was built using the Evolutionary Algorithm described in section III to define the pruning model. Our depth first search was given a seed snake of length 37 from a maximal length snake in dimension 8, which can be found in [11]. The seed was necessary to provide a search space that could be searched exhaustively. An exhaustive search with pruning from the 38th node yielded five new snakes of length 188. One of these snakes, in transition sequence notation:

```

0 1 2 3 4 5 3 2 1 0 3 2 5 3 1 2 3 4 5 2 3 1 0 3 2 6 0 5
4 3 0 1 3 5 0 2 3 1 0 3 5 4 3 0 1 3 7 4 1 3 0 1 5 2 0 1 3 0
5 4 1 3 0 6 2 1 0 3 4 5 3 0 1 3 2 0 5 3 1 0 3 4 5 2 6 3 0 2
6 1 2 3 6 7 8 6 3 2 0 5 2 1 0 3 4 5 3 0 1 3 2 0 5 3 1 0 3 4
5 2 6 3 0 2 5 3 4 5 0 3 1 0 2 5 1 0 3 1 2 4 7 6 5 2 3 1 0 3
6 2 0 3 1 0 2 5 6 2 1 3 4 5 0 3 1 0 2 3 5 6 2 5 0 2 1 3 0 2
5 0 1 2 6 1 3 4 6 3

```

This constitutes an improvement over the previous lower bound of 186.

B. Dimension 10

Because the solution space for Q_{10} is exponentially bigger than that of Q_9 , we were forced to constrain the solution space more harshly. The pruning model was defined using only 30 snakes found by the Evolutionary Algorithm, and we used the first 185 nodes of the best known Q_9 snake (listed above) to seed the depth first search. It should be noted that our search of the constrained solution space was not exhaustive. After two weeks of runtime, two snakes of length 363 were found (the search proceeded for several more weeks without termination or improvement). One of these snakes, in transition sequence notation:

```

0 1 2 3 4 5 3 2 1 0 3 2 5 3 1 2 3 4 5 2 3 1 0 3 2 6 0 5
4 3 0 1 3 5 0 2 3 1 0 3 5 4 3 0 1 3 7 4 1 3 0 1 5 2 0 1 3 0
5 4 1 3 0 6 2 1 0 3 4 5 3 0 1 3 2 0 5 3 1 0 3 4 5 2 6 3 0 2
6 1 2 3 6 7 8 6 3 2 0 5 2 1 0 3 4 5 3 0 1 3 2 0 5 3 1 0 3 4
5 2 6 3 0 2 5 3 4 5 0 3 1 0 2 5 1 0 3 1 2 4 7 6 5 2 3 1 0 3
6 2 0 3 1 0 2 5 1 0 3 6 4 3 1 6 2 1 0 5 2 0 3 1 2 0 5 2 6 5
3 2 0 1 3 0 5 4 3 1 9 5 3 4 0 2 3 1 0 2 5 0 3 4 5 2 6 3 0 2
5 3 4 5 0 3 1 0 2 5 1 0 3 1 4 3 6 1 0 3 1 8 4 5 3 2 0 1 3 2
5 3 1 6 5 3 2 0 5 3 1 5 2 3 5 4 3 1 5 2 0 1 5 6 0 3 5 2 3 0
1 3 2 5 6 7 3 6 2 1 0 3 4 5 3 0 1 3 5 6 1 3 0 1 5 2 0 1 3 0
5 4 3 5 6 8 2 0 1 3 2 5 0 1 4 5 0 3 1 0 2 3 5 6 2 5 0 2 1 3
0 2 5 0 1 2 6 1 3 4 6 3 0 1 5 2 0 1 3 0 2 8 0 6 2 5 4 3 0 1
3 5 4 2 3

```

This constitutes an improvement over the previous lower bound of 358.

VI. CONCLUSION

We intend to expand the search technique to the hypercubes of higher dimensions. In addition, we shall adapt the search to be applied to the problem of finding lengthy coils. We have no reason to believe that the search will not work equally well on these problems, and hope that further lower bounds will be established.

It is likely that our method would benefit from further pruning heuristics. The three described in this paper are simply those which seemed most relevant, and more informative measures may well exist.

REFERENCES

- [1] H.L. Abbott and M. Katchalski, "On the snake in the Box Problem", *Journal of Combinatorial Theory*, vol. 45, pp. 13–24, 1988.
- [2] Abbott, H.L. and M. Katchalski, "On the Construction of Snake-in-the-Box Codes". *Utilitas Mathematica* vol. 40, pp. 97–116, 1991.
- [3] D.A. Casella and W.D. Potter, "New Lower Bounds for the Snake-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Snakes", *Proc. Florida Artificial Intelligence Research Society Conference*, 2005.
- [4] D.W. Davies, "Longest 'separated' paths and loops in an N cube". *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 261, 1965.
- [5] F. Harary, J.P. Hayes and H.J. Wu, "A Survey of the Theory of Hypercube Graphs". *Computational Mathematics Applications*, vol. 40, pp. 277–289, 1988.
- [6] M. Juric, W.D. Potter and M. Plaskin, "Using PVM for Hunting Snake-in-the-Box codes" *Proc. 1994 Transputer Research and Applications Conference*, pp. 97–102, 1994.
- [7] W.H. Kautz, "Unit-Distance Error-Checking Codes" *IRE Trans. Electronic Computers*, vol. 7, pp. 179–180, 1980.
- [8] V. Klee, "What is the Maximum Length of a d-Dimensional Snake?." *American Mathematics Monthly*, vol. 77, pp. 63–65, 1970.
- [9] K.J. Kochut, "Snake-in-the-Box codes for Dimension 7." *Journal of Combinatorial Mathematics and Combinatorial Computations*, vol. 20, pp. 175–185, 1998.
- [10] W.D. Potter, R.W. Robinson, J.A. Miller and K.J. Kochut. "Using the Genetic Algorithm to find Snake-in-the-Box Codes", *Proc. Proceedings of the 7th International Conference On Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 421–426, 1994.
- [11] D.S. Rajan and A.M. Shende, "Maximal and Reversible Snakes in Hypercubes", Presented at the *Proc. 24th Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computation*, 1999.
- [12] H.S. Snevily, "The snake-in-the-box problem: A new upper bound", *Discrete Mathematics*, vol. 133, pp. 307–314, 1994.
- [13] J. Wojciechowski, "On the length of snakes in powers of complete graphs", *J. London Math. Soc.*, vol. 71, pp. 22–32, 2005.