4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp1-6)

New Lower Bounds for the Coil-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Coils

D. A. CASELLA and W. D. POTTER
Artificial Intelligence Center
111 GSRC, Univ. of Georgia
Athens, Georgia 30602-7415
USA

Abstract: - The coil-in-the-box problem is a variation of a difficult problem in mathematics and computer science, known as the snake-in-the-box problem, that was first described by Kautz in the late 1950's [7]. Coil-in-the-box codes have many applications in electrical engineering, coding theory, and computer network topologies. Generally, the longer the coil for a given dimension, the more useful it is in these applications [9]. By applying a relatively recent evolutionary search algorithm known as a population-based stochastic hill-climber, new lower bounds were achieved for the longest-known coil in each of the dimensions nine through eleven.

Key-Words: - Coil-in-the-Box, Snake-in-the-Box, Circuit Codes, Snake, Coil, Genetic Algorithm

1 Introduction

Hunting for 'coils,' or achordal induced cycles, in an n-dimensional hypercube deals with finding the longest cycle, following the edges of the hypercube, that obeys certain constraints. First, every two nodes, or vertices that are consecutive in the cycle must be adjacent to each other in the hypercube, second, every two nodes that are not consecutive in the cycle must not be adjacent to each other in the hypercube [5]. Snakes, like coils, are paths through the hypercube, but unlike coils these paths are open, not closed.

An n-dimensional hypercube contains 2n nodes that can be represented by the 2n n-tuples of binary digits of the hypercube's Gray code [5]. By labeling each node of the hypercube with its Gray code representation, adjacencies between nodes can be detected by the fact that their binary representations differ by exactly one coordinate. Using this fact, one can immediately detect if any two nodes in the hypercube are adjacent by performing the logical exclusive-or, hereafter referred to as XOR, operation on them and confirming that the result is an integer power of two. Using this information, a strategy for detecting, as well as generating, node adjacencies can be generalized to any dimension. In Figure 1, the bold line segment illustrates a cycle through a three-dimensional hypercube. This cycle, {0, 1, 3, 7, 6, 4, 0} in integer representation and {000, 001, 011, 111, 110, 100, 000} in binary representation, is one specific example of a longest-possible coil for a three-dimensional hypercube. The length of this coil is six, as the length of a coil always refers to the number of transitions, or edges, in the cycle.

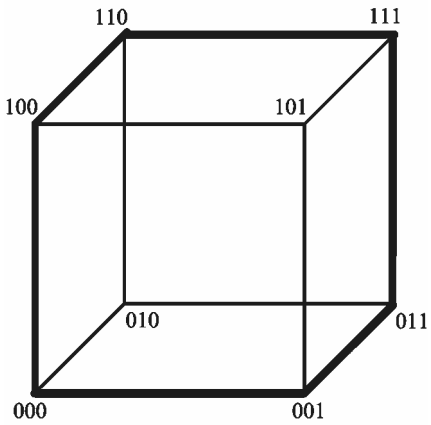

Figure 1: 3-D hypercube with embedded coil.

2 Previous Coil-Hunting Approaches

Traditionally, mathematical approaches to the coil-in-the-box, hereafter referred to as CIB, problem have involved two fundamental strategies. The first is a method of construction utilizing the tools of logic, discrete mathematics, and graph theory, while the second uses mathematical analysis to reduce the search-space followed by a computationally exhaustive search of the remaining, more-tractable,

search-space. These techniques have been successful in determining the longest-possible coils for hypercubes of dimensions one through six as shown in Table 1. For dimension seven, the lower bound for longest-possible snake [13] was found independently using both a genetic algorithm and an exhaustive search, while the lower bound for longest-possible coil [10] was found using only an exhaustive search.

DIM	SNAKE	COIL
1	1	0
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.

But even as mathematical approaches to solving this problem have been strengthened through the use of computational techniques, the combinatorial explosion in the size of the search-space as the dimension number increases has become a barrier for these methods. For dimensions eight and above, even with currently available hardware, an exhaustive search remains impractical. This has opened the door for less-traditional, heuristic-based computational search techniques. One increasingly popular branch of these search techniques is known as stochastic search algorithms. This area includes stochastic hill-climbers, tabu search, simulated annealing, evolutionary strategies, genetic algorithms, and hybrids of these particular types such as memetic algorithms.

One example of a stochastic search algorithm that has been used to hunt for snakes and coils in dimensions seven and eight is the genetic algorithm, hereafter referred to as the GA. Developed by John Holland [6], the GA is based on the simulation of Darwinian evolution and uses an evolutionary loop composed of fitness-based selection of individuals from a population, crossover of these individuals' genetic material, and mutation of these individuals' genes. The GA performs a search-space reduction through the use of a heuristic in determining the fitness of an individual within the population and through the inherent parallelism of a population-based approach. This technique has met with success and, by finding, what were at that time, record-breaking snakes in dimensions seven and eight [13], proved its effectiveness for snake hunting.

Another stochastic search algorithm that has proven successful in traveling salesperson-type problems, such as the CIB problem, is the stochastic hill-climber [8]. A population-based stochastic hill-climber, hereafter referred to as PBSHC, is very similar in structure and operation to the simple GA with a few key differences in the evolutionary cycle. The first difference is the absence of a crossover operator. Where the GA models sexual reproduction, the PBSHC models asexual reproduction in that the children in each generation are created directly from the parents of the previous generation without the exchange of genetic material between them. The second difference is the addition of a growth operator. The growth operator is the component that does most of the actual 'hill-climbing' as it chooses randomly from the nodes available to extend each snake's path by one edge along the hypercube. The absence of crossover, broadly considered the most important operator of a GA, can sometimes leave the PBSHC at a relative disadvantage. However, due to the trouble most crossover schemes have in dealing with adjacencies, the lack of crossover did not seriously degrade the PBSHC's performance, relative to the GA, in the case of the SIB problem.

3 The PBSHC Evolutionary Cycle

Hunting for coils is a much more difficult problem than hunting for snakes due to the added constraint of having to be a closed path. Two approaches were attempted in our effort to evolve populations of coils. The first approach was made using an operator that allowed each coil in the population to replace a random node with three nodes that maintained a valid coil while becoming two edges longer in the process. A second approach used a previously developed algorithm for the evolution of snakes by allowing the snakes to become coils, recording them, and then removing them from the population. The second approach, which proved to be much more effective, was subsequently chosen as the method we used to hunt for coils in dimensions nine through twelve and is described in this section.

Each individual in the population consists of a sequence of integers that represents the node sequence of a snake, or valid open 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 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 a node, other than the end node, that is already in the snake. The choice of tightness as a component of the fitness function was inspired by the idea that tighter snakes might tend to be longer snakes. Since all snakes that were able to grow in the previous generation have the same length, 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 selection based upon each snake's fitness. Three fundamental selection types were considered. Preliminary trials were conducted using roulette-wheel, tournament, and rank-based selection methods. Roulette-wheel, or probabilistic selection, proved the least effective, and most computationally expensive. Tournament selection was more effective than roulette-wheel at producing longer snakes, on average, over multiple runs. Rank-based selection out-performed both roulette-wheel and tournament selection, by maintaining a more diverse population throughout the evolutionary cycle. In rank-based selection, after first ranking the population by fitness, selection takes place based on a set percentage of the population.

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. Modifications were made to the growth operator in order to grow coils from the population of snakes. This modification allows for the selection of end nodes that are adjacent to the current snake's start node, forming a coil. When this happens, the coil can no longer be extended under this algorithm. In order to keep the entire population eligible for extension, the fitness function was modified to record any coils found, and then set their fitness to zero so that they would be replaced with snakes in the next generation, allowing the snake-evolving algorithm to function properly. The result of these modifications is that the same effective operators could be used to hunt for both snakes and coils, simultaneously. Unidirectional growth was chosen for implementation. 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: typically, enhancing the best individual in a population is a standard approach used in hybrid genetic algorithms [12]). 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 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.

After growth, the mutation operator acts on each snake in the population. The mutation operator we experimented with is an enhancement of a basic XOR mutation scheme [2]. In the standard XOR mutation scheme, a node is chosen at random from within the snake, excluding the start node and the end node. The chosen node's neighboring nodes are XOR'd and that result is then XOR'd with the chosen node in order to exchange it with a different node that maintains local adjacency requirements with the original node's neighbors. The enhancement of this operator is referred to as iterative-conditional XOR mutation and is somewhat more computationally expensive, but also more effective. Instead of choosing a node at random, each node in the snake, with the exception of the start node and the end node, is mutated and tested for any improvement in fitness within a copy of the original snake. If any improvement is found, that mutation is added to a mutation pool. Upon testing of all nodes within the snake, the snake of best fitness from the mutation pool is substituted for the original snake. If any improvement was found through mutation, the entire process is repeated until no further improvement is found. This scheme ensures that only constructive mutation is allowed, and that undergoing mutation can never reduce an

individual's fitness. By making this mutation conditional, it has also become a 'hill-climbing' component of the evolutionary cycle. However, left to iterate without bounds this scheme may lead to prohibitive run-times. Restricting the number of iterations for each original snake to five for example keeps the advantages of a local hill-climber and also keeps the runtime under control. It turns out that this enhancement did not influence our results because the operator was turned off in order to prevent changes in the high-quality root-snakes used when seeding results from lower to higher dimensions.

4 The PBSHC Parameter Settings

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 and PBSHC 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.

5 Results

The best results to date were achieved using populations of ten thousand, a selection percentage of ninety percent, and by seeding each population, with snakes, at startup. While the goal of these particular runs was to hunt for coils in particular, due to the implementation of the PBSHC's operators, coils cannot be used for seeding the population. Using a technique where the best snakes found in each dimension were used as seeds for the next higher dimension [13], the PBSHC was able to find coils longer than the previously known longest coils for 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 [14] 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 each dimension were used as seeds for the next dimension's runs. In conjunction with this technique, mutation was shut off in order to preserve the high-quality seeds generated from each previous dimension. This also reduced the runtime allowing larger populations to be run over a much shorter time-frame.

DIM	PREVIOUS BEST	PBSHC
8	96	90
9	170	**180**
10	340	**344**
11	620	**630**
12	1238	1236

Table 2: Comparison of old and new lower bounds for coils.

In Table 2, for dimension eight, the lower bound for longest-known coil [1] was found using traditional mathematical proof and construction techniques. The previous best-known lower bounds were 170 for dimension nine [11], and 340, 620, and 1238 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for longest-known coils in dimensions nine through eleven are listed here.

dim 9 : 180

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

6 5 0 2 5 3 1 2 5 0 2 3 5 4 3 2 1 5 2 0 1 3 2 0 6 4 0 2
3 1 0 5 4 3 0 1 3 2 1

dim 10 : 344

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

dim 11 : 630

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 2 5 0 2 3 7 2 0 5 4 2
1 3 0 1 5 2 0 1 3 0 5 4 3 5 2 0 3 6 2 5 4 3 0 1 3 5 0 2
3 1 0 3 5 4 3 0 1 2 5 0 2 8 7 3 2 4 5 3 1 0 3 4 5 3 1 5
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 2 0
3 1 7 6 2 5 4 3 5 1 3 2 5 3 0 1 3 5 6 2 5 4 3 5 0 1 3 5
4 3 0 5 2 0 1 5 4 1 0 3 5 6 2 3 5 0 9 5 1 0 7 8 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 5 0 8 1 6 2 3 0 1 3 2 5 4 3 5 1 3 2
5 3 0 1 5 3 6 0 3 1 0 2 5 0 3 4 2 3 0 1 2 0 7 3 4 2 1 3
0 1 5 2 0 1 3 0 5 4 3 5 2 0 3 6 5 0 4 5 3 2 5 1 3 5 0 2
3 5 4 0 8 2 1 3 4 5 0 3 1 0 2 3 5 0 2 4 5 2 3 1 0 3 6 2
0 5 2 3 1 4 2 1 0 5 2 0 3 1 2 0 5 4 2 8 5 6 10 4 3 5 8
2 1 5 2 3 6 2 5 4 3 5 2 0 3 6 2 5 4 2 6 1 2 4 3 6 5 0 3
4 0 2 3 6 9 1 0 2 5 0 3 4 5 3 1 0 5 3 4 5 2 7 3 2 0 1 3
2 9 3 0 2 5 0 1 6 2 1 0 9 7 0 1 2 3 4 5 2 3 1 0 3 6 5 3
0 1 3 8 1 9 2 4 3 0 5 2 0 3 6 2 3 1 5 3 2 9 5 3 1 0 3 5
4 3 0 1 2 5 1 3 5 4 3 0 1 6 5 0 1 3 0 5 4 3 5 2 0 1 9 6
1 0 2 3 6 2 7 4 9 3 0 1 3 2 0 5 2 7 9 2 5 0 4 9 0 2 7 3
9 4 2 5 6 3 5 0 2 3 1 0 2 9 0 3 2 5 3 1 2 3 4 5 2 3 1 0
3 2 6 0 3 1 5 0 6 5 8 1 6 2 0 1 3 2 5 4 2 6 5 2 0 1 7 5
4 7 1 9 0 5 2 3 1 0 3 2 6 0 3 1 0 5 2 1 0 3 6 8 5 2 3 1
5 2 6 4 3 2 4 0 1 6 3 8 0 9

6 Conclusion

The results of using a PBSHC to search for longest constrained paths in multi-dimensional hypercubes, as demonstrated by our current and previous [3, 4] work are very encouraging and will, hopefully, lead to further acceptance and application of hybrid evolutionary techniques within the research community. As computational hardware improves over time, these technique should prove useful in even higher dimensions both in this particular problem and also to other related problems in computational graph theory. Further improvements using this implementation of the PBSHC are anticipated including the return to bi-directional growth, the addition of a mid-coil growth operator, and a parallel virtual machine implementation of the PBSHC.

7 Acknowledgements

We would like to thank the students and staff of the AI Center, especially Dr. Michael Covington, for their support and cooperation with this project. There were numerous days and nights when the Center's lab computers were occupied with coil hunting.

References:

[1] Abbott, H. L. and M. Katchalski. 1991. On the Construction of Snake-In-The-Box Codes. *Utilitas Mathematica* 40:97-116.

[2] Brown, W. April, 2004. Personal Communication.

[3] Casella, D. A., W. D. Potter. New Lower Bounds for the Snake-in-the-box Problem: Using Evolutionary Techniques to Hunt for Snakes. *Proceedings of the 18th International Florida Artificial Intelligence Research Seminar,* 264-269. Clearwater Beach, Florida.

[4] Casella, D. A., W. D. Potter. Using Evolutionary Techniques to Hunt for Snakes and Coils. *Proceedings for 2005 Congress on Evolutionary Computation,* 2499-2504. Edinburgh, Scotland.

[5] Harary, F., J. P. Hayes, and H. J. Wu. 1988. A Survey of the Theory of Hypercube Graphs. *Computational Mathematics Applications* 15:277-289.

[6] Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press.

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

[8] Kingdon, J. and L. Dekker. 1995. The Shape of Space. In *Proceedings of the Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 543-548. London, UK.: IEE.

[9] Klee, V. 1970. What is the Maximum Length of a d-Dimensional Snake? *American Mathematics Monthly* 77:63-65.

[10] Kochut, K. J. 1996. Snake-In-The-Box Codes for Dimension 7. *Journal of Combinatorial Mathematics and Combinatorial Computations* 20:175-185.

[11] Paterson, K. G. and J. Tuliani. 1998. Some New Circuit Codes. *IEEE Transactions on Information Theory* 44(3):1305-1309.

[12] Potter, W. D., J.A. Miller, B.E. Tonn, R.V. Gandham, and C.N. Lapena. 1992. Improving The Reliability of Heuristic Multiple Fault Diagnosis Via The Environmental Conditioning Operator. *APPLIED INTELLIGENCE: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies* 2: 5-23.

[13] Potter, W. D., R. W. Robinson, J. A. Miller, and K. J. Kochut. 1994. Using the Genetic Algorithm to Find Snake-In-The-Box Codes. In *7th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, 421-426. Austin, Texas.

[14] Rajan, D. S. and A. M. Shende. 1999. Maximal and Reversible Snakes in Hypercubes. Presented at the *24th Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computation*.