New Lower Bounds for the Snake-in-the-Box and the Coil-in-the-Box Problems: Using Evolutionary Techniques to Hunt for Snakes and Coils

by

Darren A. Casella

(Under the direction of Walter D. Potter)

Abstract

The snake-in-the-box problem is a difficult problem in mathematics and computer science that deals with finding the longest-possible constrained path that can be formed by following the edges of a multi-dimensional hypercube. This problem was first described by Kautz in the late 1950's (Kautz 1958). Snake-in-the-box codes, or 'snakes,' are open paths while coil-in-the-box codes, or 'coils,' are closed paths, or cycles. Snakes and coils have many applications in electrical engineering, coding theory, and computer network topologies. Generally, the longer the snake or coil for a given dimension, the more useful it is in these applications (Klee 1970). By applying a relatively recent evolutionary search algorithm known as a population-based stochastic hill-climber, new lower bounds were achieved for (1) the longest-known snake in each of the dimensions nine through twelve and (2) the longest-known coil in each of the dimensions nine through eleven.

Index Words: Snake-in-the-Box, Coil-in-the-Box, Circuit Codes, Snake, Coil, Genetic Algorithm, Population-Based Stochastic Hill-Climber

New Lower Bounds for the Snake-in-the-Box and the Coil-in-the-Box Problems: Using Evolutionary Techniques to Hunt for Snakes and Coils

by

Darren A. Casella

B.S., Florida State University, 1997

M.A., Arizona State University, 2003

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2005

New Lower Bounds for the Snake-in-the-Box and the Coil-in-the-Box Problems: Using Evolutionary Techniques to Hunt for Snakes and Coils

by

Darren A. Casella

Approved:

Major Professor: Walter D. Potter

Committee: Khaled Rasheed
O. Bradley Bassler

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2005

Dedication

This thesis is dedicated to my family,

especially:

my mother for teaching me to read,

my father for endless trips to the library,

Alicia for always being there with encouragement,

and to the memory of my teacher and friend Melissa Prevatt (1943-2002).

Acknowledgements

The faculty, staff, and students of the AI Center have created a wonderful atmosphere of support and encouragement. I would like to thank everyone who helped make this thesis possible, especially, Dr. Potter for his enthusiasm towards this project, his ideas on making it work, and his efforts to always make it better, Dr. Rasheed for suggesting an expansion of the search to higher dimensions, without which none of these new lower bounds would have been found, and Dr. Bassler for answering so many of my questions on how the ideas of quantum theory will change our understanding of the theory of computation and always asking interesting questions. I would also like to thank Dr. Covington for drawing the three-dimensional hypercubes that made these papers so instantly recognizable. And last, but far from least, I would like to thank all my friends at the AI Center for making it fun to spend time in the lab.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this research was to determine if there are ways to customize the implementation of a simple genetic algorithm in order to improve its performance within a specific, well-researched domain. The domain chosen for this research is a branch of graph theory that focuses on searching for both open paths, known as 'snakes,' and closed paths, known as 'coils,' within multi-dimensional hypercubes. This problem has been widely researched in discrete mathematics and graph theory for over fifty years. Much progress has been made over this time by traditional 'proof-based' approaches, but more recently new results have been discovered using techniques which make use of computational artificial intelligence. The most successful of these computational approaches seems to be the simple genetic algorithm.

This thesis describes, over the course of three papers, the results of research into customizing the genetic algorithm to hunt for snakes and coils in hypercubes of dimensions nine through twelve. This research grew out of an assignment in the Computational Intelligence course taught at the AI Center. Upon the completion of the initial assignment, the snake-hunting algorithm was a simple genetic algorithm capable of using probabilistic, tournament, or rank-based selection, enhanced edge-recombinative crossover, and random XOR-mutation on a population of snakes in node-based representation. Each chromosome could contain multiple snakes within it and the longest within each chromosome would be detected and used to calculate that chromosome's fitness. The fitness function for this early version was based only on the length of the snake. While the results of using this algorithm to hunt for snakes in dimension eight were somewhat successful, finding a snake of length 89, I became obsessed with the idea that there must be some way to improve its performance. Before the

official deadline of the assignment, the first steps of the algorithm's customization were taken. Crossover was eliminated and the crossover operator was replaced with a growth operator that allowed one end of the snake to grow to an adjacent node in the hypercube. In this early version, the growth operator was naive and allowed snakes in the population to grow to nodes that violated global adjacencies, resulting in their elimination. With just these two early modifications, snakes of length 90 and 91 were found in dimension eight. Convinced that other improvements in the algorithm could be found, I decided to spend the summer of 2004 in the AI Center's computer lab, hunting for snakes.

The next, and one of the most important, changes of the snake-hunting algorithm, came with a new fitness function. Many different ways were attempted to incorporate the idea of 'tightness' into the fitness function, but eventually it was realized that due to the divergence of the multi-objective function of fitness between length and tightness, one of them must be allowed to dominate. The choice was made to allow length to dominate tightness within the fitness function. This choice required further changes to ensure the entire population would always have the same length during the selection process. To accommodate this new requirement, we made changes to the chromosomal representation. Though still an integer array, instead of allowing snakes to exist anywhere in the chromosome and performing a computation to find the snake within the chromosome, the new representation has the zero node as its first element and, by only allowing the snake to grow to adjacent nodes that do not violate global adjacencies, maintains a single, valid snake throughout the evolutionary process. At the same time, the original XOR-mutation scheme was also modified to a conditional XOR-mutation. In this new form, instead of picking a random node to mutate, a number of copies of the snake are made and a different index is XOR-mutated for each snake. If any of these mutations result in an improved fitness, then the best of them is used to replace the original snake in the population. The results of these changes were very promising and snakes through length 96 were found in dimension eight.

After all these changes, however, the algorithm had still not produced a snake of length 97, which is the longest-known snake for dimension eight. After discussions with my committee about the failure to find any length-97 snakes, it was decided to generalize the algorithm to hunt for snakes in higher dimensions. After modification, runs were undertaken in dimensions nine through twelve, using the best results of each dimension to seed the next-higher dimension's runs. Within a period of weeks, new lower bounds for the longest-known snake were broken in each of these dimensions. Further review of the literature revealed a straightforward technique to convert a snake hunter into a coil-hunter and trials in this more-difficult venture were begun. Over the following months new lower bounds for longest-known coils were found for dimensions nine through eleven. The current incarnation of this snake/coil-hunting algorithm has evolved into a population-based stochastic hill climber and, as the following results will attest, has proven to be an extremely competent snake and coil hunting technique.

Chapter 2

New Lower Bounds for the Snake-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Snakes[1]

[1]Casella, D. A., and W. D. Potter. 2005. Accepted by the *18th International Florida Artificial Intelligence Research Seminar, FLAIRS '05.* Reprinted here with permission of publisher.

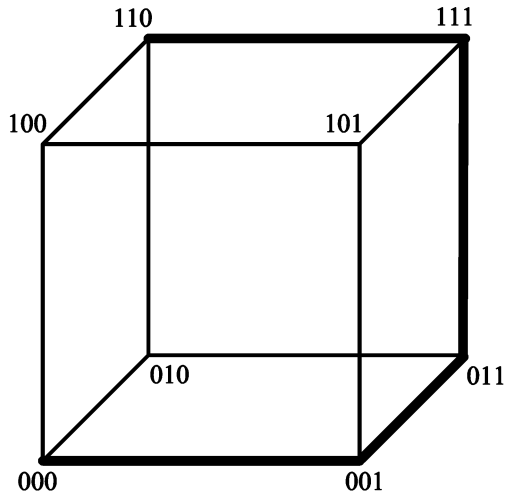

Figure 2.1: Three-dimensional hypercube with embedded snake.

2.1 What is a Snake?

Hunting for 'snakes,' or achordal induced paths, in an n-dimensional hypercube deals with finding the longest path, following the edges of the hypercube, that obeys certain constraints. First, every two nodes, or vertices that are consecutive in the path must be adjacent to each other in the hypercube, and second, every two nodes that are not consecutive in the path must not be adjacent to each other in the hypercube. A third constraint, whether the path is open or closed, determines if the path is a 'snake' or a 'coil.' While coils have received the most attention in the literature (Harary, Hayes, and Wu 1988), both snakes and coils have useful applications. This paper deals primarily with searching for open paths, or snakes, where the start node is not adjacent to the end node.

An n-dimensional hypercube contains 2^n nodes that can be represented by the 2^n n-tuples of binary digits of the hypercube's Gray code. As illustrated in Figure 2.1, 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

Table 2.1: Maximum lengths of snakes and coils.

Dim	**Snake**	**Coil**
1	1	0
2	2	4
3	4	6
4	7	8
5	13	14
6	26	26
7	50	48

(Harary, Hayes, and Wu 1988). 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 2.1, the bold line segment represents an open path, or snake, through the three-dimensional hypercube. This path, {0, 1, 3, 7, 6} in integer representation and {000, 001, 011, 111, 110} in binary representation, is one specific example of a longest-possible snake for a three-dimensional hypercube. The length of this snake is four, as the length of a snake always refers to the number of transitions, or edges, in the path.

Table 2.1 shows the maximum lengths of snakes and coils for dimensions one through seven. These values have all been proven, through exhaustive search, to be the longest-possible snakes and coils for their respective dimension.

2.2 Previous Snake-Hunting Approaches

Traditionally, mathematical approaches to the snake-in-the-box, hereafter referred to as SIB, problem have involved two fundamental strategies. The first is a method of construction uti-

lizing the tools of logic, discrete mathematics, and graph theory, while the second makes use of mathematical analysis to reduce the search-space followed by a computationally exhaustive search of the remaining, tractable, search-space. These techniques have been successful in determining the longest-possible snakes for hypercubes of dimensions one through six as shown in Table 2.1. For dimension seven, the lower bound for longest-possible snake (Potter et. al. 1994) was found independently using both a genetic algorithm and an exhaustive search, while the lower bound for longest-possible coil (Kochut 1996) was found using only an exhaustive search.

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 in dimension eight is the genetic algorithm, hereafter referred to as the GA. Developed by John Holland (Holland 1975), 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 (Potter et. al. 1994), proved its effectiveness for snake hunting.

Another stochastic search algorithm that has proven successful in traveling salesperson-type problems, such as the SIB problem, is the stochastic hill-climber (Kingdon and Dekker 1995). 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 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 the GA, can sometimes leave the PBSHC at a relative disadvantage. However, due to the trouble most crossover schemes have in dealing with global adjacencies, the lack of crossover did not seriously degrade the PBSHC's performance, relative to the GA, in the case of the SIB problem.

2.3 Evolutionary Cycle of the Snake-Hunting PBSHC

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 of the snake 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 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 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 an adjacent node in the hypercube that has not been disqualified by already being in the snake, or by being adjacent to some node that is in the snake. Both unidirectional and bi-directional growth were implemented, with bi-directional growth allowing each snake to grow from either end. 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 (Potter et. al. 1992)). 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 works well in conjunction with the fitness function, but does require 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 (Brown 2004). 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 in this scheme to five or less 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.

2.4 Parameter Settings for the Snake-Hunting PBSHC

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 being modified to run in dimensions nine through twelve. Our previous experience with genetic algorithm snake hunters supported the application of lower dimension parameter settings to higher dimensions. 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 these experiments 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.

2.5 Results of the Snake-Hunting PBSHC

The best results to date were achieved using population sizes of ten thousand, a selection percentage of ninety percent, and by seeding each population at startup. Using a technique where the best snakes found in each dimension were used as seeds for the next higher dimension (Potter et. al. 1994), the PBSHC was able to find snakes longer than the previously known longest snakes for dimensions nine through twelve, as highlighted in Table 2.2. However, to generate good seeds for dimension nine, a bootstrap solution was used. This method involved cutting a dimension-eight, length-97 snake (Rajan and Shende 1999) back to its

Table 2.2: Comparison of old and new lower bounds for snakes.

Dim	Previous Best	PBSHC
8	97	97
9	168	**186**
10	338	**358**
11	618	**680**
12	1236	**1260**

length-50 root, corresponding to a longest-possible 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.

In Table 2.2, for dimension eight, the lower bound for longest-known snake (Rajan and Shende 1999) was found using traditional mathematical proof and construction techniques aided by computational search (i.e., construction based on lower dimension long snake characteristics). For dimensions nine through twelve, the previous best-known lower bounds were derived by calculation from the lower bounds for longest-known coils in each dimension. Any coil can be converted to a snake by removing one node. This results in a snake whose length is two less than the original coil, as removing one node removes the two edges connecting it within the coil. The previous best-known lower bounds were 168 for dimension nine (Abbott and Katchalski 1991), and 338, 618, and 1236 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for longest-known snakes in dimensions nine through twelve are listed in Appendix A.

2.6 Conclusions

The results of using a PBSHC to search for SIB codes are very encouraging. As computational hardware improves over time, this technique should prove useful in even higher dimensions. Further improvements using this implementation of the PBSHC are anticipated including the return to bi-directional growth, the addition of a mid-snake growth operator, and a parallel virtual machine implementation of the PBSHC. Preliminary trials using this technique to search for coils, or closed paths, have begun and we have already discovered new lower bounds for coils in dimensions nine (180), ten (344), and eleven (630); details to be reported in a future paper.

2.7 References

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

Brown, W. April, 2004. Personal Communication.

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

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

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

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.

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

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

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

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.

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.

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.*

Chapter 3

New Lower Bounds for the Coil-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Coils[1]

[1]Casella, D. A., and W. D. Potter. Submitted to the *WSEAS/IASME International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, CIMMACS '05.*

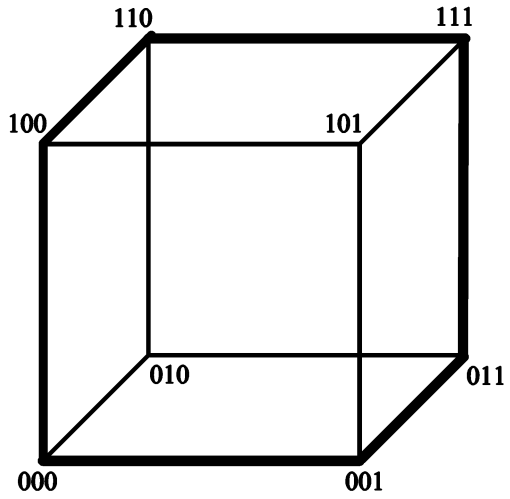

Figure 3.1: Three-dimensional hypercube with embedded coil.

3.1 What is a Coil?

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 (Harary, Hayes, and Wu 1988). Closed paths, or cycles satisfying these two constraints are known as 'coils,' while open paths satisfying them are known as 'snakes.'

An n-dimensional hypercube contains 2^n nodes that can be represented by the 2^n n-tuples of binary digits of the hypercube's Gray code. 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 (Harary, Hayes, and Wu 1988). 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 3.1, the bold line segment illustrates a cycle, or coil, through the 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 coil.

3.2 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 makes use of 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 (Potter et. al. 1994) was found independently using both a genetic algorithm and an exhaustive search, while the lower bound for longest-possible coil (Kochut 1996) was found using only an exhaustive search.

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, sim-

ulated 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 (Holland 1975), 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 (Potter et. al. 1994), 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 (Kingdon and Dekker 1995). 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 the GA, can sometimes leave the PBSHC at a relative disadvantage. However, due to the trouble most crossover schemes have in dealing with global adjacencies, the lack of crossover did not seriously degrade the PBSHC's performance, relative to the GA, in the case of the CIB problem.

3.3 Evolutionary Cycle of the Coil-Hunting PBSHC

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 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 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 an adjacent node in the hypercube 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 (Potter et. al. 1992)). 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 (Brown 2004). 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 in this scheme to five or less 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.

3.4 Parameter Settings for the Coil-Hunting PBSHC

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 being 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.

3.5 Results of the Coil-Hunting PBSHC

The best results to date were achieved using population sizes 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 (Potter et. al. 1994), the PBSHC was able to find coils longer than the previously known longest coils for dimensions nine through eleven, as highlighted in Table

Table 3.1: Comparison of old and new lower bounds for coils.

Dim	**Previous Best**	**PBSHC**
8	96	90
9	170	**180**
10	340	**344**
11	620	**630**
12	1238	1236

3.1. In order to generate good seeds for dimension nine, a bootstrap solution was used. This method involved cutting a dimension-eight, length-97 snake (Rajan and Shende 1999) 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.

In Table 3.1, for dimension eight, the lower bound for longest-known coil (Abbott and Katchalski 1991) was found using traditional mathematical proof and construction techniques. The previous best-known lower bounds were 170 for dimension nine (Abbott and Katchalski 1991), 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 in Appendix B.

3.6 Conclusions

The results of using a PBSHC to search for CIB codes are very encouraging. As computational hardware improves over time, this technique should prove useful in even higher dimensions. 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. Previous trials using this technique to search for snakes, or open paths, have already discovered new lower bounds for snakes in dimensions nine (180), ten (344), eleven (630), and twelve (1260) as reported in our previous paper (Casella and Potter 2005).

3.7 References

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

Brown, W. April, 2004. Personal Communication.

Casella, D. A. and W. D. Potter. 2005. New Lower Bounds for the Snake-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Snakes. Accepted by the *18th International Florida Artificial Intelligence Research Seminar*. Clearwater Beach, Florida.

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

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

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

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.

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

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

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

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.

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.

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.*

Chapter 4

Using Evolutionary Techniques to Hunt for Snakes and Coils[1]

[1]Casella, D. A., and W. D. Potter. Submitted to the *2005 IEEE Congress on Evolutionary Computation, CEC2005.*

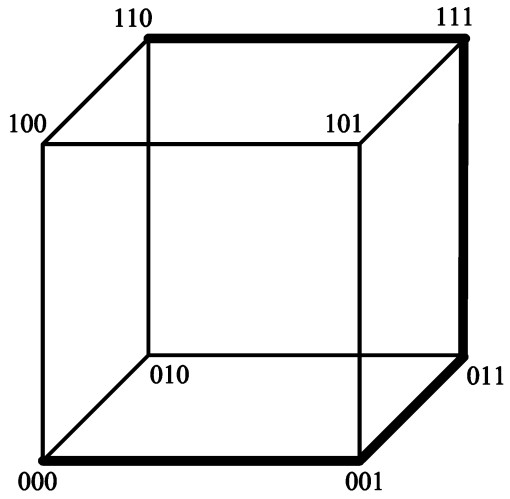

Figure 4.1: Three-dimensional hypercube with embedded snake.

4.1 Snake and Coils in Multi-Dimensional Hypercubes

Hunting for 'snakes' and 'coils' in an n-dimensional hypercube deals with finding the longest path, following the edges of the hypercube, that obeys certain constraints. First, every two nodes, or vertices, that are consecutive in the path must be adjacent to each other in the hypercube, and second, every two nodes that are not consecutive in the path must not be adjacent to each other in the hypercube. A third constraint, whether the path is open or closed, determines if the path is a snake or a coil. While coils have received the most attention in the literature (Harary, Hayes, and Wu 1988), both snakes and coils have useful applications. This paper details our attempt to hunt for both snakes and coils using evolutionary search algorithms.

An n-dimensional hypercube contains 2^n nodes that can be represented by the 2^n n-tuples of binary digits of the hypercube's Gray code as illustrated in Figures 4.1 and 4.2. 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

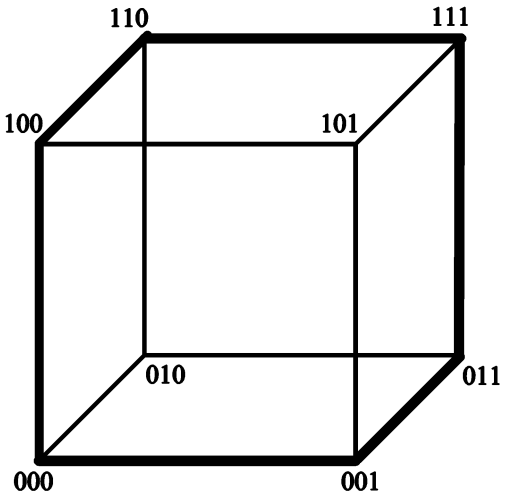

Figure 4.2: Three-dimensional hypercube with embedded coil.

coordinate (Harary, Hayes, and Wu 1988). 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 4.1, the bold line segment illustrates an open path, or snake, through a three-dimensional hypercube. This path, {0, 1, 3, 7, 6} in integer representation and {000, 001, 011, 111, 110} in binary representation, is one specific example of a longest-possible snake for a three-dimensional hypercube. The length of this snake is four, as the length of a snake always refers to the number of transitions, or edges, in the path.

In Figure 4.2, the bold line segment illustrates a closed path, or coil, through a three-dimensional hypercube. This path, {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.

Each of these paths can also be represented in the form of a transition sequence, such as {0, 1, 2, 0} for the snake and {0, 1, 2, 0, 1, 2} for the coil. Here each number in the sequence is the coordinate of the Gray code bit which is being changed to generate the next node of the snake or coil. Transition sequence representations can start from either zero or one, but in this paper we follow the convention of starting with zero as it both seems more intuitive and simplifies the mathematics when converting between node sequences and transition sequences. All snakes and coils listed in this paper will follow the form of transition sequence representation in order to conserve as much space as possible.

4.2 Previous Approaches to Hunting for Snake and Coils

Traditionally, mathematical approaches to the snake-in-the-box, as well as the coil-in-the-box problems, hereafter referred to collectively as the path-in-the-box, or PIB, 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 snakes and coils for hypercubes of dimensions one through six as shown in Table 1. For dimension seven, the lower bound for longest-possible snake (Potter et. al. 1994) was found independently using both a genetic algorithm and an exhaustive search, while the lower bound for longest-possible coil (Kochut 1996) was found using only an exhaustive search.

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 (Holland 1975), 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 the individuals' genetic material, and mutation of the 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 dimension seven and eight (Potter et. al. 1994), proved its effectiveness for snake hunting.

Another stochastic search algorithm that has proven successful in traveling salesperson-type problems, such as the PIB problem, is the stochastic hill-climber (Kingdon and Dekker 1995). 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 the GA, can sometimes leave the PBSHC at a relative disadvantage. However, due to the trouble most crossover schemes have in dealing with global adjacencies, the lack

of crossover did not seriously degrade the PBSHC's performance, relative to the GA, in the case of the PIB problem.

4.3 Evolutionary Cycle of the PBSHC

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 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.

Initial attempts of integrating both length and tightness into a fitness function met with limited success. Many combination of linear and non-linear factors of length and tightness for the fitness function were tried. While they each would perform 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 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, a predetermined percentage of the population is selected to carry over to the next generation. 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 an adjacent node in the hypercube that has not been disqualified by already being in the snake, or by being adjacent to some node that is in the snake. A modification was made to the growth operator in order to grow coils as well as open-path snakes. This modification allows for the selection of end nodes that are adjacent to the current snake's start node, resulting in the formation of 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 also modified to record any coils found, and then replace them with snakes of zero fitness that will be replaced in the next generation. The result of these modifications is that the same effective operators

could be used to hunt for both snakes and coils, simultaneously. 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: typically, enhancing the best individual in a population is a standard approach used in hybrid genetic algorithms (Potter et. al. 1992)). 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.

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 (Brown 2004). 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 logically 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 in this scheme to five or less 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.

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 the 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 the snake-hunting algorithm and then allowed the snakes to become coils by permitting the snakes to grow to nodes that were adjacent to the start node, recorded the resulting coils, and then removed 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.

4.4 Parameter Settings for the PBSHC

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 being modified to run in dimensions nine through twelve. Our previous experience with genetic algorithm

Table 4.1: Comparison of old and new lower bounds for snakes and coils.

Dim	Previous Best Snake	SHPBSHC	Previous Best Coil	CHPBSHC
8	97	97	96	90
9	168	**186**	170	**180**
10	338	**358**	340	**344**
11	618	**680**	620	**630**
12	1236	**1260**	1238	1236

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.

4.5 Results of Hunting Snakes and Coils with the PBSHC

The best results to date were achieved using population sizes of ten thousand, a selection percentage of ninety percent, and by seeding each population with highly-fit, lower-dimension snakes at startup. In Table 4.1, a comparison is made between the previously-known lower

bounds for snakes and coils and the results achieved using the snake-hunting and coil-hunting PBSHCs. Using a technique where the best snakes found in each dimension were used as seeds for the next higher dimension (Potter et. al. 1994), 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, as highlighted in Table 4.1.

In order to generate good seeds for dimension nine, a bootstrap solution was used. This method involved cutting a dimension-eight, length-97 snake (Rajan and Shende 1999) 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.

Although the PBSHC performed well in hunting for both snakes and coils, it failed to match or raise the lower bounds on coils for dimension eight or twelve, demonstrating that hunting for coils is an inherently more difficult problem than hunting for snakes. One factor to consider at this point is that by starting each run with lower-dimension seeds of high fitness means that the search space is restricted to branches having those seeds as their root, eliminating the huge number of potentially long snakes that do not extend from that branch. This limitation may be the reason that the PBSHC could not match or improve upon the lower bounds for coils in dimensions eight and twelve.

For dimension eight, the lower bound for longest-known snake (Rajan and Shende 1999) was found using traditional mathematical proof and construction techniques aided by computational search (i.e., construction based on lower dimension long snake characteristics). For dimensions nine through twelve, the previous best-known lower bounds were derived by calculation from the lower bounds for longest-known coils in each dimension. Any coil can

be converted to a snake by removing one node. This results in a snake whose length is two less than the original coil, as removing one node removes the two edges connecting it within the coil. The previous best-known lower bounds were 168 for dimension nine (Abbott and Katchalski 1991), and 338, 618, and 1236 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for snakes in dimensions nine through twelve are listed in Appendix A.

For dimension eight, the lower bound for longest-known coil (Abbott and Katchalski 1991) was found using traditional mathematical proof and construction techniques. The previous best-known lower bounds were 170 for dimension nine (Abbott and Katchalski 1991), and 340, 620, and 1238 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for coils in dimensions nine through eleven are listed in Appendix B.

4.6 Conclusions

The results of using a PBSHC to search for PIB codes are very encouraging. As computational hardware improves over time, this technique should prove useful in even higher dimensions. Further improvements using this implementation of the PBSHC are currently being considered. We anticipate that this technique would work well as a distributed process, with each thread searching a specific sub-region of the hypercube for a linear factor increase in search speed.

4.7 References

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

Brown, W. April, 2004. Personal Communication.

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

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

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

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.

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

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

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

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.

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.

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.*

Chapter 5

Conclusion

By finding new lower bounds for (1) the longest-known snakes in each of the dimensions nine through twelve, and (2) the longest-known coils in each of the dimensions nine through eleven, we have shown that there are, indeed, benefits to modifying stochastic search techniques to their particular domain of application. These results also testify to the effectiveness of hybrid approaches that can combine the strengths of different strategies. While the population-based stochastic hill-climber, by strict definition, is no longer a simple genetic algorithm, the modifications necessary for the transformation from the GA to the PBSHC were identified by realizing which genetic operators were not ideally matched to the specific problem domain and finding suitable replacements. Once this step had taken place, the remaining changes were undertaken to tailor the architecture of the evolutionary cycle for maximum efficiency.

Appendix A

Transition Sequences Of New Lower Bound Snakes

A.1 dim 9 : 186

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 2 4 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

A.2 dim 10 : 358

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

A.3 dim 11 : 680

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

A.4 DIM 12 : 1260

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 3 5 2 6 3 0 2 5 3
4 5 0 3 1 0 2 5 1 0 3 1 5 6 2 0 5 3 1 5 2 3 5 4 3 1 0 4 7 1 3 5 2 3 1 0 2 3 5 4 0 8 3 1 0 3 4 5 3
0 1 3 2 0 5 3 1 0 3 4 5 3 1 0 2 3 7 2 0 5 4 2 0 3 5 2 0 1 3 0 5 4 3 5 2 0 1 6 4 1 0 2 5 1 3 5 2 7
3 5 1 0 3 1 2 3 5 1 3 0 5 4 3 5 2 0 3 1 2 8 4 0 5 4 3 2 1 0 3 2 4 3 0 5 2 0 1 4 6 1 0 2 3 6 2 1 7
5 8 1 5 2 0 3 5 1 8 3 10 9 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 7 4 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 9
4 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 8 9 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 9 6 0 2 3 1 0 2 5 3 4 5 0 3 1 0 2 3 5 6 2 3
1 5 2 3 4 5 3 0 1 3 8 4 1 3 0 1 5 3 7 2 5 3 1 5 2 0 1 5 4 1 2 5 3 0 2 6 3 2 5 3 4 5 0 3 1 0 2 5 1
0 3 4 1 0 8 1 4 5 3 2 5 1 3 5 0 2 3 5 4 1 2 9 0 3 4 5 0 3 1 0 2 5 1 0 3 1 2 4 5 2 3 1 0 3 6 2 0 5
4 3 0 5 2 0 3 1 2 0 5 4 9 1 4 5 0 3 1 5 3 2 1 3 0 1 4 9 0 2 5 0 9 1 2 0 3 6 2 0 11 5 0 9 4 3 0 5

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

Appendix B

Transition Sequences Of New Lower Bound Coils

B.1 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

B.2 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

B.3 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