

# Hunting for Snakes in Hypercubes using Genetic Algorithms

Pedro A. Diaz-Gomez<sup>a</sup> and Dean F. Hougen<sup>a</sup>  
 University of Oklahoma, U.S.A

## Abstract

Hunting for snakes of maximum length in hypercubes has been addressed with non-heuristic methods for hypercubes of dimensions less than eight. Above that dimension the problem is intractable because the search grows exponentially with the dimension, which makes it an NP-hard problem. Heuristic methods, like genetic algorithms, have been used to solve this kind of problem. This article evaluates different fitness functions to find snakes in hypercubes of dimension greater than three, compares their performance with different initial populations and mutation rates, conjectures an upper bound on the maximum number of points of longest snakes, and poses some open questions regarding the number of maximum length snakes in hypercubes.

**Keywords:** Genetic algorithms, snakes, fitness function, mutation operator, initial population.

## 1. Introduction

Longest snakes in hypercubes are of interest in coding theory [20], digital design, and telecommunications [9]. A  $d$ -dimensional hypercube is a connected, non-directed graph of  $2^d$  nodes, where each node has  $d$  neighbors and a binary labeling of each node may be given that differs in exactly one bit from that of each of its neighbors [14]. Figure 1 shows a 4-dimensional hypercube with such a labeling.

A *snake* is a complete, connected, open path in the  $d$ -hypercube where each node belonging to the path has two neighbors, except the head and the tail which have only one neighbor each. That is, a node in the snake is adjacent to at most two nodes in the path and there must be exactly two distinguished nodes, the head and the tail, each with only one neighbor in the path [2]. If the connected path is such that it has no head and tail, i.e., it is a cycle where each node belonging to the path has exactly two neighbors in the path, then the path is called a *coil* [2]<sup>1</sup>. Figure 1 shows a path that is a longest snake in a 4-dimensional hypercube<sup>2</sup>.

<sup>a</sup>Robotics, Evolution, Adaptation, and Learning Laboratory  
 School of Computer Science  
 Norman, OK 73019, USA  
 pdiazg@ou.edu, hougen@ou.edu

<sup>1</sup>Previously some mathematicians have referred to a coil as a snake ([12] and personal communication from H. S. Snevily, 2006).

<sup>2</sup>While the binary labeling hints at the reason snakes are of interest in fields such as coding theory, we will use a base-10 labeling in the remaining figures in this document, due to the increased familiarity of the decimal system.

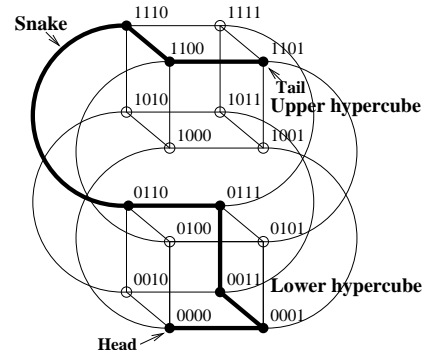


Fig. 1. Hypercube of dimension 4 with a longest possible snake (length 7).

The problem of finding longest snakes in hypercubes is a search problem in  $2^d$ -dimensional space, which is of order  $O(2^{2^d})$  when the snake is encoded as a binary array. This makes the problem of finding longest snakes in hypercubes an appealing one for heuristic methods like genetic algorithms [19].

The problem of longest snakes in hypercubes is so pronounced that the length of longest snakes is unknown for hypercubes of dimensions greater than 7. Different conjectures about upper and lower bounds have been proposed for coils [1], [7], [12], [21]. For snakes a lower bound has been proposed [5] and empirical studies have tried to achieve those bounds [2], [5], [19], [20]. This article proposes a conjecture for the upper bound on the length of snakes and reports enhancements of previous work [4], [5] evaluating seven fitness functions with different populations, different mutation rates, and different stop criterion in 4-dimensional hypercubes.

The organization of this article is as follows: Section 2 introduces how the snake is encoded and evaluates previous fitness functions [4], [5] in hypercubes of dimension 4; Section 3 analyzes the effectiveness of the fitness functions during the first 1,000 generations, and when the stop criterion is finding the longest snake; besides that, it shows two methods of evaluating the length of a snake; Section 4 evaluates the performance of the fitness functions when the initial population is changed; Section 5 addresses the impact of the mutation operator on the reduction of the number of generations for solving the problem; Section 6 conjectures an upper bound in the length of longest snakes

	Adjacency Matrix									
Node	0	1	2	3	4	5	6	7	<i>S</i>	<i>VN</i>
0	0	1	1	0	1	0	0	0	1	<u>1</u>
1	1	0	0	1	0	1	0	0	1	<u>2</u>
2	1	0	0	1	0	0	1	0	0	<u>3</u>
3	0	1	1	0	0	0	0	1	1	<u>2</u>
4	1	0	0	0	0	1	1	0	0	<u>2</u>
5	0	1	0	1	0	0	0	1	0	<u>2</u>
6	0	0	1	0	1	0	0	1	1	<u>1</u>
7	0	0	0	1	0	1	1	0	1	<u>2</u>

Table I. Encoding for hypercube and candidate snake in three dimensions including adjacency matrix. The underlined values in the Vector of Neighbors (*VN*) correspond to nodes in the candidate snake (*S*).

and posts some open questions; and Section 7 is the conclusions and future work.

## 2. Genetic Algorithms to Hunt Snakes

A *genetic algorithm* is a parallel search method inspired by biological evolution [10]. Usually an initial set of random possible solutions is generated. Then the candidate solutions are evaluated with a *fitness function* [15]. The best solutions are probabilistically chosen to go to the next generation without change or to give rise to new offspring with the application of operators such as crossover and mutation.

To solve a problem with genetic algorithms, one must encode the problem in order to find the solution. Here the snake is encoded as an unidimensional array (chromosome) of length  $2^d$ , where  $d$  is the dimension of the hypercube in which to search for snakes. In this chromosome, each 1 means that the corresponding node belongs to the snake and each 0 means that node does not belong to the snake. The hypercube is encoded as an adjacency-matrix. To determine the number of neighbors of each node in a candidate solution array, the matrix multiplication between the adjacency-matrix (the hypercube) and the array (the hypothesized snake) is performed; this results in the Vector of Neighbors (*VN*) [4], [5]—see Table I.

Once the representation has been chosen, the problem is how to evaluate the candidate solutions in order to choose the best ones found so far so as to improve on them to find a snake of the maximum possible length. Here the fitness function must be taken into account; it is going to evaluate candidate solutions throughout the entire process.

Besides the influence of the fitness function on the quality of the solution, factors such as initial population membership, population size, selection mechanism(s), crossover and mutation probabilities, and stop criteria influence the performance of the algorithm in terms of the quality of the obtained solution and/or the number of generations to obtain it. This article focuses on changing the fitness function, the initial population membership, and

the mutation rate, and looks at the impact of those changes on the performance of the algorithm. This does not mean that the other parameters are not important but that we have chosen just these for the present. Elsewhere we study the impact of the crossover rate on performance for this type of problem [6].

### 2.1. Fitness Functions

To determine the effects of various approaches to constructing fitness functions for this problem, we conduct an empirical study of seven fitness functions. The experimental settings for this study are: initial population randomly generated, population size 10 individuals<sup>3</sup>, chromosome size  $2^4$ , one point crossover with probability 60%, mutation probability 3% per chromosome, 2-tournament selection with a 75% chance of selecting the more fit individual and a 25% chance of selecting the less fit individual, and 1,000 generations maximum [4].

Fitness functions usually join objective(s) with constraint(s) [3]. The *objective* here is to find a longest snake, i.e., to maximize the number of nodes in the path. The *constraints* are to ensure that no characteristic of a snake is violated, i.e., to ensure that: (1) the number of neighbors of each node belonging to the path is no greater than 2, and (2) the path has exactly two distinguished points with one neighbor each.

For the first constraint it should be noted that “belonging to the path” means that the nodes (points) in the unidimensional array that represents the snake form a connected path. There should not be *isolated points*, i.e., points marked as a 1 in the array, but unconnected—see Figure 2.

The second constraint takes into account whether there are no such distinguished points, or there is only one, or there are more than 2.

In Figure 2 there is a chromosome 110100000010010 with the path 0 – 1 – 3 – 11, one isolated point (14), and one *lazy point* (13, which is a point with no neighbors in the path). However if point 15 is added, the isolated and lazy points disappear and give rise to a snake 0 – 1 – 3 – 11 – 15 – 14.

Following the normal guidelines for constructing fitness functions [3], one needs to determine how to join the objective—maximum length—and the constraints—retaining the properties of snakes.

*2.1.1. A Normalized Fitness Function:* As a first approach to joining the objective and the constraints, consider

<sup>3</sup>Only ten because the search space is  $2^{2^4}$ , which is quite small.

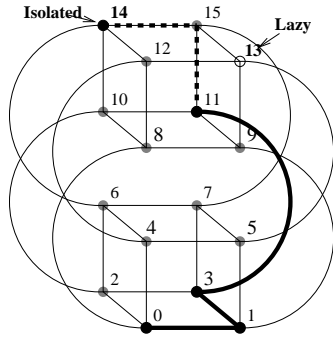


Fig. 2. Isolated and lazy points in a 4-dimensional hypercube.

Run	Equation 1						Equation 2					
	Fit.	Len.	#I.	#L.	#B.	#D.	Fit.	Len.	#I.	#L.	#B.	#D.
1	1.00	6	0	0	0	2	5.81	6	1	0	0	2
2	1.00	6	0	0	0	2	7.00	7	0	0	0	2
3	1.00	6	0	0	0	2	6.00	6	0	0	0	2
4	1.00	6	0	0	0	2	5.81	6	1	0	0	2
5	1.00	6	0	0	0	2	6.00	6	0	0	0	2
6	0.96	6	0	1	0	2	7.00	7	0	0	0	2
7	1.00	6	0	0	0	2	7.00	7	0	0	0	2
8	1.00	7	0	0	0	2	5.81	6	1	0	0	2
9	1.00	5	0	0	0	2	5.81	6	1	0	0	2
10	1.00	6	0	0	0	2	5.81	6	1	0	0	2
Totals												
10	9.96	60	0	1	0	20	62.05	63	5	0	0	20

Table II. Results with fitness functions as in Equations 1 and 2.

the normalized fitness function given in Equation 1 [5]:

$$F(I) = \left( \frac{\sum_{j=0}^{2^d-1} VN_j - Penalty}{\sum_{j=0}^{2^d-1} VN_j} \right) \frac{Length(S) + 1}{\#P} \quad (1)$$

where  $VN$  is the vector of number of neighbors,  $Penalty$  corresponds to the violation of the constraint (i.e., the count of number of points belonging to the path which have more than 2 neighbors, plus the absolute value of the difference between the number of distinguished points and 2, plus the number of lazy points),  $Length(S)$  corresponds to the length of the connected path from one distinguished point to the other or until the constraint is violated, and  $\#P$  is the total number of points in the chromosome. Whenever  $F(I)$  in Equation 1 is equal to 1 there is a snake in the corresponding hypercube.

The results on 10 runs are given in the first part of Table II, where *Run* is the number of the corresponding run, *Fit.* is the maximum value of the fitness in that run, *Len.* is the maximum length of the snake found in that run, *#I.* is the number of isolated points found in that chromosome, *#L.* is the number of lazy points in that chromosome, *#B.* is the number of bad points (which are those that have more than two neighbors), and *#D.* is the number of distinguished points (which must be equal to 2 in a snake).

With the normalized fitness function as in Equation 1,

on 10 runs the following can be observed from Table II: (1) The fitness function reaches the maximum value (1.0) in 90% of the cases. (2) In all cases the algorithm finds snakes, including the case where the fitness value is 0.96, because the lazy point does not violate the constraint, i.e., lazy points are not good for finding longer snakes but they can occur. (3) The algorithm finds a longest snake in run 8. (4) The algorithm has no way to differentiate based on the lengths of the snakes, i.e., a snake of length 6 is as fit as a snake of length 7 because the fitness value is 1.0 in both cases. (5) In all runs the algorithm finds the two distinguished points.

**2.1.2. A Length-Differential Fitness Function:** In order to see if the fitness function can usefully differentiate between snakes of different lengths, and to see if the number of longest snakes can be improved, Equation 1 is changed slightly.

If the second factor of Equation 1 is changed so that the length of the snake found is no longer normalized by the number of points in the array then the equation becomes

$$F(I) = \left( \frac{\sum_{j=0}^{2^d-1} VN_j - Penalty}{\sum_{j=0}^{2^d-1} VN_j} \right) * Length(S). \quad (2)$$

If the same type of test is performed as in Subsection 2.1.1, taking into account that  $0 \leq Length(S) \leq 7$ , then we get the following results—see Table II, second part: (1) The fitness function reaches maximum a fitness value (7.0) in 30% of the cases. (2) In 50% of the cases the algorithm finds snakes; in the other 50% the graph of the chromosome is unconnected—to get a snake the isolated point must be removed. (3) The algorithm finds a longest snake on 3 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In all runs the algorithm finds the two distinguished points.

**2.1.3. A Single-Length-Dependent Fitness Function:** The length of the snake is a good distinguishing factor for snakes, so consider that factor alone as a fitness function.

$$F(I) = Length(S). \quad (3)$$

One may think that perhaps no constraint is present in Equation 3 but this is not the case because when the length of the snake is calculated we begin with a distinguished point—head or tail—and follow the path until the constraint is violated. The results obtained now are quite similar to the ones found in Section 2.1.2, but there are differences as well—see Table III, first part: (1) The fitness function reaches a maximum (7) in 50% of the cases. (2) In 90% of the cases the algorithm finds snakes; in the other 10% the graph of the chromosome is unconnected. (3) The algorithm finds a longest snake on 5 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In all runs the algorithm finds the two distinguished points.

Run	Equation 3						Equation 4					
	Fit.	Len.	#I.	#L.	#B.	#D.	Fit.	Len.	#I.	#L.	#B.	#D.
1	7	7	0	0	0	2	48	6	1	0	0	2
2	7	7	0	0	0	2	56	7	0	0	0	2
3	7	7	0	0	0	2	56	7	0	0	0	2
4	6	6	0	1	0	2	48	6	1	0	0	2
5	6	6	0	1	0	2	13	1	0	0	9	1
6	6	6	0	1	0	2	56	7	0	0	0	2
7	6	6	0	1	0	2	56	7	0	0	0	2
8	7	7	0	0	0	2	48	6	1	0	0	2
9	7	7	0	0	0	2	56	7	0	0	0	2
10	6	6	1	0	0	2	48	6	1	0	0	2
Totals												
10	65	65	1	4	0	20	485	60	4	0	9	19

Table III. Results with fitness functions as in Equations 3 and 4.

2.1.4. *A Quadratic Fitness Function:* It may be useful to penalize candidate solutions containing lazy points. Consider a fitness function that is quadratic in the number of points in the chromosome:

$$F(I) = (\#P - \#Lazy) * Length(S). \quad (4)$$

This function is called quadratic because as  $Length(S)$  is a function of the number of points, Equation 4 is quadratic in the number of points ( $\#P$ ). Results are shown in Table III, second part.

This time something interesting happens: There is a chromosome with a fitness value of 13 and “snake” length of 1 with 9 bad points in it—see Table III, second part. This happened because of the factor  $(\#P - \#Lazy)$  in Equation 4. Additionally: (1) The fitness function reaches a maximum (56) in 50% of the cases. (2) In 50% of the cases the algorithm finds snakes; in the other 50% the graph of the chromosome is unconnected. (3) When the algorithm finds snakes, those found are longest, i.e., the algorithm finds a longest snake on 5 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In 90% of the runs the algorithm finds the two distinguished points.

2.1.5. *A Linear Fitness Function:* Consider a fitness function similar to Equation 3 with a penalty for lazy points:

$$F(I) = Length(S) - \#Lazy. \quad (5)$$

Now the results are shown in Table IV, first part, including: (1) The fitness function reaches a maximum (7) in 70% of the cases. (2) In 90% of the cases the algorithm finds snakes; in the other 10% the graph of the chromosome is unconnected. (3) The algorithm finds a longest snake on 7 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In all runs the algorithm finds the two distinguished points.

Success at finding longer snakes improves by 20% compared to results using Equation 3 and 4, and all

Run	Equation 5						Equation 6					
	Fit.	Len.	#I.	#L.	#B.	#D.	Fit.	Len.	#I.	#L.	#B.	#D.
1	6	6	0	0	0	2	6	6	0	0	0	2
2	7	7	0	0	0	2	7	7	0	0	0	2
3	6	6	1	0	0	2	6	6	0	1	0	2
4	7	7	0	0	0	2	7	7	0	0	0	2
5	7	7	0	0	0	2	7	7	0	0	0	2
6	7	7	0	0	0	2	7	7	0	0	0	2
7	6	6	0	0	0	2	6	6	1	0	0	2
8	7	7	0	0	0	2	6	6	0	0	0	2
9	7	7	0	0	0	2	7	7	0	0	0	2
10	7	7	0	0	0	2	7	7	0	0	0	2
Totals												
10	67	67	1	0	0	20	66	66	1	1	0	20

Table IV. Results with fitness functions as in Equations 5 and 6.

chromosomes show snakes except one—see Table IV, run 3, where there is an isolated point.

2.1.6. *A Linear Fitness Function Taking into Account Lazy Points and Isolated Points:* As good results are obtained with a previous linear function, consider a new linear function that takes into account both lazy and isolated points:

$$F(I) = Length(S) - \#Lazy - \#Isolated. \quad (6)$$

Results are shown in Table IV, second part, including: (1) The fitness function reaches a maximum (7) in 60% of the cases. (2) In 90% of the cases the algorithm finds snakes; in the other 10% the graph of the chromosome is unconnected. (3) The algorithm finds a longest snake on 6 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In all runs the algorithm finds the two distinguished points. Contrary to what may be expected, we do not obtain better results than with Equation 5 because this time 6 longest snakes are obtained vs. 7 as is shown in Table IV, first and second part, column *Len.*. The new term  $\#Isolated$  in Equation 6 does not fix it at all, as shown in entry 7 of the same Table, second part.

2.1.7. *A Rational Fitness Function:* As the objective is to find longest snakes, two factors should be considered: (1) longest, which means with a maximum number of points, and (2) snakes, which is the constraint—see Section 2.1. This introduces the idea of a *rational fitness function*, i.e., a fraction where the numerator is the objective and the denominator is the constraint. In this way if the numerator increases, then the fraction increases, and if the denominator decreases, then the fraction increases, accomplishing the goal of maximizing the fitness function. The fitness function proposed is then

$$F(I) = \frac{Length(S)}{1 + Penalty} \quad (7)$$

where *Penalty* is as defined in Section 2.1.1.

Run	Fit.	Len.	#I.	#L.	#B.	#D.
1	7.0	7	0	0	0	2
2	6.0	6	0	0	0	2
3	6.0	6	0	0	0	2
4	7.0	7	0	0	0	2
5	3.0	6	0	1	0	2
6	6.0	6	0	0	0	2
7	3.0	6	0	1	0	2
8	3.0	6	0	1	0	2
9	3.0	6	0	1	0	2
10	7.0	7	0	0	0	2
Totals						
10	51.0	63	0	4	0	20

Table V. Results with fitness function as in Equation 7.

Results are shown in Table V, including: (1) The fitness function reaches a maximum (7.0) in 30% of the cases. (2) In all cases the algorithm finds snakes. (3) The algorithm finds a longest snake on 3 runs. (4) The algorithm has a way to differentiate lengths of snakes. (5) In all runs the algorithm finds the two distinguished points. In conclusion, all final chromosomes are snakes with this fitness function and it finds longest snakes 30% more of the time compared to Equation 1 in Section 2.1.1—but this time there are 4 lazy points in the 10 runs.

### 3. Effectiveness of Fitness Functions in Finding Longest Snakes in 4-Dimensional Hypercubes

Fitness functions have been evaluated for hunting snakes in 4-dimensional hypercubes from Equation 1 to Equation 7. Results are summarized in Table VI, where the tests were made over 30 runs to obtain more statistically interesting results. The best fitness functions for finding snakes are Equations 1, 6, and 7 but, of the three, Equation 6 finds more of the maximum possible length. Equation 4 has the peculiarity that almost all the snakes that it finds are longest ones. All these results are obtained when the algorithm is stopped at 1,000 generations.

Table VII shows t-test results for these equations,

Eq.	# Sn.	# Long.	T.Length	# T.I.	# T.L.	# T.B.	# T.D.
1	30	2	182	0	5	0	60
2	15	7	97	15	0	0	60
3	26	19	175	4	5	0	60
4	17	15	134	11	0	14	59
5	24	18	168	6	0	0	60
6	28	21	189	2	3	0	60
7	28	16	184	2	4	0	20

Table VI. Comparing results with different fitness functions and a maximum of 1,000 generations. 30 runs per fitness function. For each fitness function # Sn. is the total number of snakes found during all 30 runs, # Long. is the number of longest snakes found, T.Length is the total length of all snakes found, # T.I. is the total number of isolated points remaining in the fittest individual at the end of the run, # T.L. is the total number of such lazy points, # T.B. is the total number of such bad points, and # T.D. is the total number of distinguished points.

Eq.	1,000 Iterations						Gen. until Longest Found					
	1	2	3	4	5	6	1	2	3	4	5	6
2	.00	-	-	-	-	-	.81	-	-	-	-	-
3	.05	.01	-	-	-	-	.91	.85	-	-	-	-
4	.01	.03	.35	-	-	-	.67	.89	.69	-	-	-
5	.03	.01	.95	.39	-	-	.08	.12	.05	.06	-	-
6	.08	.00	.50	.11	.33	-	.12	.12	.076	.11	.88	-
7	.01	.03	.67	.67	.71	.18	.78	.95	.83	.95	.13	.17

Table VII. Unpaired t-test results for longest snakes. Probability of results assuming the null hypothesis. 30 runs per fitness function.

Equation	Number of Generations on 30 Runs			
	Minimum	Maximum	Average	$\sigma$
1	26	17,032	2,874.2	4,104.0
2	6	25,352	3,174.1	5,414.7
3	2	14,065	2,970.8	3,679.6
4	6	23,077	3,356.4	5,312.9
5	3	11,504	1,346.4	2,198.3
6	2	10,362	1,451.7	2,793.2
7	14	30,974	3,264.2	6,179.3

Table VIII. Number of generations for different fitness functions in finding longest snakes in a 4-dimensional hypercube.

comparing the length of the longest snake found in the final population after 1,000 generations. Basically Equations 1 and 2 are statistically significantly different from most of the others and from each other as well. Equations 3 and 5 are quite similar; effectively the only difference is the subtrahend #Lazy in Equation 5. Somehow the same happens between Equation 3 and 6 which has a probability value of 0.50 and where the difference is the subtrahend #Lazy - #Isolated. However this subtrahend makes Equation 6 more effective in finding snakes and longer ones than the others<sup>4</sup>.

But how would the results turn out if the algorithm is freed to find a longest snake, i.e., when the stopping criterion is the finding of a longest snake rather than stopping at 1,000 generations. Table VIII shows the minimum, maximum, average (mean), and standard deviation  $\sigma$  using this revised stop criterion.

It can be observed that, on average, the best fitness functions for finding longer snakes are Equations 5 and 6. Both have the peculiarity that they are linear and both join the objective—the length—with the constraint—the snake. As a matter of fact, all the fitness functions presented tried to do that, but in different ways. Consider Equations 3, 5, and 6 again.

Equation 3 takes into account the length of the snake, and in doing that it checks for bad points. However, if there is a lazy point, there is no way to check for it. Equation 5 takes into account the length and additionally penalizes lazy points. According to our t-test results (see Table VII, second part) Equations 5 and 3 are statistically

<sup>4</sup>Except Equation 1 that finds more snakes, but not longer ones.

significantly different at the 95% confidence level (though just barely) when the number of generations can be greater than 1,000 despite the fact that a subtrahend is the only apparent difference. However, in the short term, i.e., when the number of generations is less or equal to 1,000 there is no statistically significant difference—see Table VII, first part. For comparison, see Figures 3 and 4. This shows the effect of the performance measure on the statistical comparison of the equations.

Equation 6 takes into account the length too and, in addition, penalizes lazy and isolated points. There is no statistically significant difference between the performance of Equations 6 and 5. However, it should be taken into account that Equation 5 is considering the isolated points indirectly. When the algorithm is looking for the length of the snake, it is looking for a connected path. However, there are cases where there is a connected path and an isolated point in the graph, but in this case there is no snake and the length is not the maximum we are looking for.

Sections 2 and 3 addressed seven fitness functions and two stop criteria for hunting snakes in hypercubes. This empirical study has maintained all sets of parameters invariant except for fitness function and stop criteria. The question now is whether there are other changes that

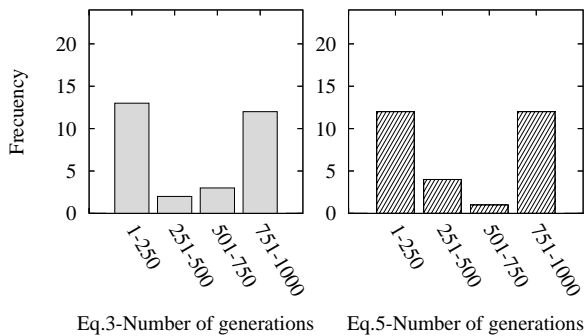


Fig. 3. Histograms for finding longest snakes using Equations 3 and 5. Number of generations until maximum fitness is reached or generations = 1,000.

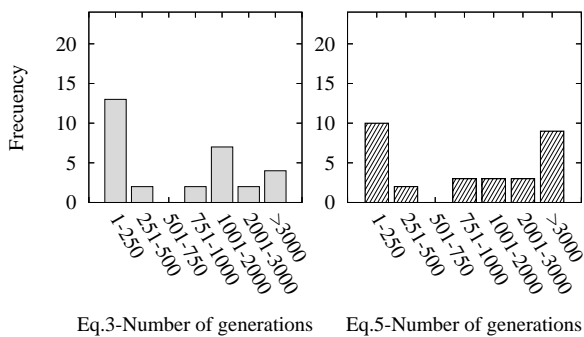


Fig. 4. Histograms for finding longest snakes using Equations 3 and 5. Number of generations until maximum fitness is reached.

improve the effectiveness of the algorithm in finding longest snakes. The following sections are going to address some of those changes.

### 3.1. Length Evaluation

The way the length of a snake is evaluated can impact the performance of the algorithm in terms of the number of generations required to find an optimum. For example, consider evaluating the length of the chromosome 01010001101000110110111010010001 in a 5-dimensional hypercube (see Figure 5).

The algorithm could start counting the length by using one of the three points: 18, 24, or 27. If the algorithm just finds one starting point (head or tail) and begins to count until the constraint is violated, then, if the algorithm begins with point 18, then the length is going to be 7 which corresponds to the path 18 – 22 – 20 – 21 – 17 – 1 – 3 – 7 because point 15 has three neighbors in the connected path, i.e., it violates the constraint. If the algorithm begins to count the length starting at point 24 then the length is going to be 3 which corresponds to the path 24 – 8 – 10 – 14. Finally, if the algorithm begins to count the length starting at point 27, the length is going to be 1 because the connected path without violation of the constraint is going to be 27–31. So, the actual chromosome could be evaluated with different lengths, depending on the starting point the

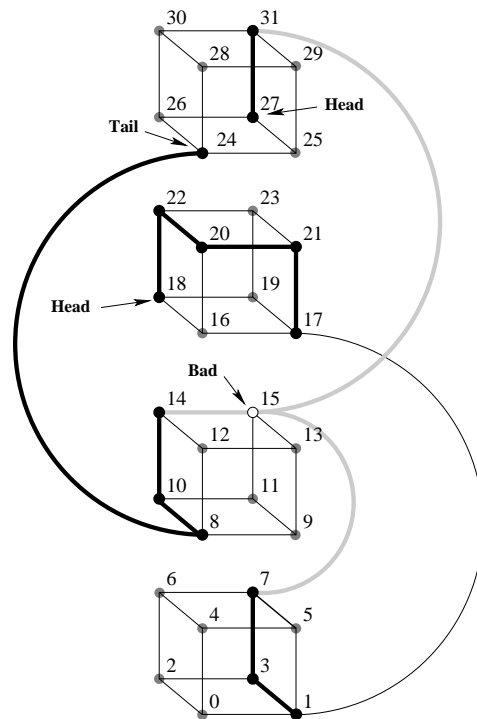


Fig. 5. Evaluation of the length of a snake.

Equation	Seed							
	0	1	2	3	4	5	6	7
1	2,874.2	719.7	1,647.4	1,578.3	1,921.6	1,002.4	798.2	2,285.3
2	3,174.1	1850.9	3,116.7	1,713.3	2,274.3	1,834.0	1,922.1	2,222.8
3	2,971.8	2208.0	2,494.7	1,685.8	1,541.0	2,194.1	2,600.8	2,083.3
4	3,356.4	3830.1	3,114.9	1,555.6	2,982.7	1,169.7	1,978.7	1,511.5
5	1,346.4	1600.5	1,799.6	3,063.9	813.4	922.9	1,947.1	1,547.8
6	1,452.7	1474.1	2,045.0	2,308.8	1,804.9	870.5	1,215.8	1,701.6
7	3,264.2	2704.9	2,621.0	2,006.5	1,573.9	1,585.2	3,086.4	1,812.4
Totals								
<i>FirstEval</i>	18,437.8	14,388.2	16,839.4	13,912.1	12,911.7	9,578.8	13,549.0	13,164.5
<i>SecondEval</i>	12,418.9	13,407.1	9,697.6	15,221.9	12,764.7	9,221.6	9,670.8	10,244.9

Table IX. Number of generations for different fitness functions in finding longest snakes in a 4-dimensional hypercube. Initial population randomly generated with different seeds. 30 runs per seed for each fitness function.

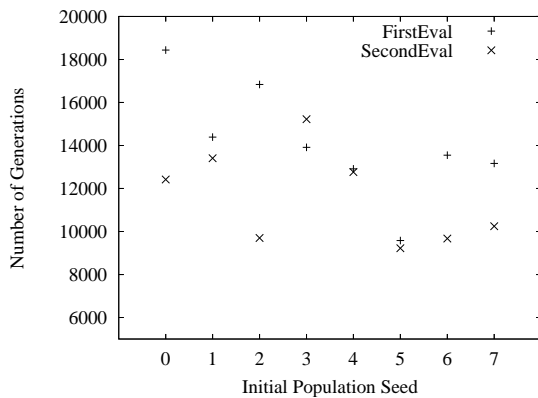


Fig. 6. Comparison of cumulative average number of generations as in Table IX ranking chromosomes in two ways: considering the first length encountered (*FirstEval*) and taking the maximum length of all encountered (*SecondEval*). Seeds values 0 through 7 shown. 30 runs per seed for each of seven fitness functions.

fitness function is using in the evaluation<sup>5</sup>. For previous tests, the evaluation was performed taking into account the first head and tail found, i.e., the ones with lower numeric node labels. Table IX shows the new totals, taking into consideration all the distinguishing points and ranking the longest connected path found that obeys the constraints—see *SecondEval* row. Figure 6 compares the two ways to conduct fitness evaluations. *FirstEval* takes the first distinguished point and finds the length; *SecondEval* takes into consideration all the connected paths that obey the constraints and ranks the chromosomes in accordance with the maximums. Eight different seeds were used to initialize the random number generator for the eight trials. With *SecondEval* there is an average reduction of 18% in the total number of generations for finding the longest snakes.

<sup>5</sup>Another possibility of evaluation is to take the sum of all lengths. However, there is not always a point like 15 in the example that makes the path connected.

#### 4. Evaluation of the Initial Population

The *initial population* is the input for the algorithm. A sufficient quantity and quality of chromosomes should produce good results [8], [11], [13], [16], [18], or at least that is what we might expect<sup>6</sup>. But besides this, how are those randomly generated chromosomes to be used by the algorithm?

Eight experiments were performed using different initial random populations of 10 individuals each, with random seeds 0 through 7. Results of the number of generations performed by a GA in finding the longest snake in a hypercube of dimension 4 are shown in Table IX, using fitness functions as in Section 2. It can be observed that for seed 0 the average number of generations needed was 18,437.8 against 9,578.8 for seed 5 (see *FirstEval* row). What makes this better performance happen?

A first analysis can be done by examining the number of points (alleles with value 1) that are initially generated in each chromosome. As genes are generated randomly, it is expected that each chromosome will have 8 zeros and 8 ones. For a hypercube of dimension 4, this is ideal because the number of points of the longest snake is 8. However, this is not the case: Seed 0 generated 5 chromosomes with 10 points, i.e., 50% of the initial population has 10 points, which may contribute to the bad performance of the algorithm when seed 0 is used; likewise, seed 7 generated 4 chromosomes with 10 points. Seed 4 has similarities with seed 7 in the sense that both have a total of 6 chromosomes with 9 or more than 9 points, 3 chromosomes with 6, 7, or 8 points, and 1 chromosome with less than 6 points. If more than 8 points could be bad, the fact that a chromosome does not have enough alleles with a value of 1 could also be bad. For example, seed 3 and seed 6 have a relatively large number of chromosomes, 2 and 3 respectively, with low numbers of points (less than six). Seed 1 and seed 5 have the peculiarity of having

<sup>6</sup>It is known that, for hunting snakes in higher dimensions, the algorithm can be seeded with a longest snake from the  $(d - 1)$ -hypercube in order to improve results [2].

Seed	Count of Number of Points				
	< 6	6 and 7	8	9	> 9
0	0	4	1	0	5
1	1	4	3	2	0
2	1	1	4	2	2
3	2	4	1	1	2
4	1	3	0	3	3
5	1	5	2	1	1
6	3	2	0	3	2
7	1	1	2	2	4

Table X. Count of number of points in chromosomes initially generated with 8 different seeds.



Fig. 7. Cumulative average performance of a GA with different fitness functions as in Table IX and different initial populations. Seed values 0 through 7 shown. 30 runs per seed for each fitness function.

the most chromosomes with the number of points equal to 6, 7, and 8; this could be good for the beginning of the algorithm. Seed 2 has the most chromosomes with 8 points (4 in this case) and was not good in performance when compared with most of the seeds. However, it should be taken into account that seed 2 has 4 chromosomes with more than 8 points, and 1 with less than 6—i.e., at least 40% of possible good chromosomes must compete with 60% of the population. This fact shows that the initial random generations of genes could be important but the initial fitness evaluation of the chromosomes should also be taken into account. Table X summarizes the initial counting of points and Figure 7 shows the cumulative average effect of the change of the initial population on all equations with respect to the number of generations necessary in finding the longest snakes, as is shown in Table IX, *FirstEval*.

### 5. The Mutation Operator

The mutation operator has not received adequate attention in evolutionary computation, in part because it is considered a background operator [17] and also because the crossover operator is usually considered the builder of possible solutions. For the 4-dimensional hypercube, mutation is an operator capable of producing a large change because with a population of size 10, once the algorithm

Eq.	$p_m$ as in Equation 8					
	1	2	3	4	5	6
2	.010	-	-	-	-	-
3	.001	.870	-	-	-	-
4	.003	.791	.901	-	-	-
5	.001	.491	.554	.658	-	-
6	.005	.517	.580	.669	.973	-
7	.065	.457	.322	.302	.151	.192

Table XII. T-test for hunting longest snakes in a 4-dimensional hypercube; different fitness functions; adaptive mutation rate; number of generations until longest snake found; Probability of results given the null hypothesis is shown; 30 runs per fitness function

converges, if it has not reached an optimum then the operator that could still teach it is mutation. The mutation probability was changed so that it could be applied in accordance with the average fitness value of the entire population.

If the mutation parameter is variable according to

$$p_m(t + 1) = \frac{\mu_{max} - \bar{\mu}(t)}{\mu_{max}} * 100\% \quad (8)$$

where  $\mu_{max}$  is the maximum fitness value and  $\bar{\mu}(t)$  is the average fitness value of the population at step  $t$  (generation  $t$ ), then Equation 8 is such that if at time step  $t$  the normalized average fitness value of the population is quite far from the optimum of 1, then  $p_m(t+1)$  is high. If, on the contrary, the average normalized fitness value of the entire population is quite near the optimum, then  $p_m(t + 1)$  is going to be low. Comparative results with a fixed mutation probability per chromosome of 3% are shown in Table XI. Table XII shows the t-test when Equation 8 is used. Equation 1 is statistically significantly different from all the others, except for Equation 7.

Table XIII compares the average number of generations in order to find a maximum when Equation 8 is used against other mutation rates: a fixed mutation rate of 3% per chromosome, as illustrated previously; mutation rates of 100% per chromosome after generation 1, 000, 500, or 250; a mutation rate equal to the generation; a mutation rate equal to the number of individuals in the population (10) multiplied by the generation. As the mutation rate increases, the number of generations needed to reach the maximum diminishes<sup>7</sup>. Some mutation rates begin fixed and reach a full rate like *S1000*. Others increase gradually like *Generation*; but Equation 8 begins high and decreases. The idea for Equation 8 came from observation in the sense that we wanted to diminish the average number of generations and “good” results were obtained from increasing the mutation rate<sup>8</sup>.

Now, consider the performance of a fitness function

<sup>7</sup>A possible cause of this is that the length of the chromosome is short (16 bits).

<sup>8</sup>The proposal and testing of mutation rates occurred in the order specified in Table XIII.

Equation	$p_m = 3\%$				$p_m$ as in Eq. 8			
	Minimum	Maximum	Average	$\sigma$	Minimum	Maximum	Average	$\sigma$
1	26	17,032	2,874.2	4,104.0	1	42	13.3	11.1
2	6	25,352	3,174.1	5,414.7	1	92	25.8	23.2
3	2	14,065	2,970.8	3,679.6	2	85	26.7	19.0
4	6	23,077	3,356.4	5,312.9	4	86	27.4	22.4
5	3	11,504	1,346.4	2,198.3	2	103	30.1	24.1
6	2	10,362	1,451.7	2,793.2	3	154	30.3	29.5
7	14	30,974	3,264.2	6,179.3	3	82	21.5	21.2

Table XI. Results for hunting longest snakes in a 4-dimensional hypercube. Different fitness functions. Mutation rates of  $p_m = 3\%$ , and  $p_m$  as in Equation 8. 30 runs per fitness function.

Equation	Mutation rate						
	Fixed 3%	$S1000$	$S500$	$S250$	Generation	$Pop*Generation$	Equation 8
1	2,874.2	744.3	394.4	194.5	189.9	62.5	13.3
2	3,174.1	670.7	381.0	211.4	141.8	42.0	25.8
3	2,970.8	607.5	322.2	181.0	164.0	34.0	26.7
4	3,356.4	500.5	382.9	215.7	172.2	46.5	27.4
5	1,346.4	551.6	329.1	237.9	167.8	40.6	30.1
6	1,451.7	696.1	379.1	235	139.9	39.7	30.3
7	3,264.2	411.7	373.2	202.5	139.2	50.4	21.5
Totals							
	18,437.8	4,182.4	2,561.9	1,478	1,114.8	315.7	175.1

Table XIII. Average number of generations for hunting longest snakes in a 4-dimensional hypercube. Different fitness functions. Different mutation rates. 30 runs.

with different mutation rates. If, for example, a comparison is made between different mutation rates using Equation 3, histograms from Figures 8 to 11 show the net effect of the mutation rate on the number of generations using that equation. For snakes in a hypercube of dimension 4, the mutation rate is like a blade that cuts the number of generations down when it is increased. When the mutation rate is constant and equal to 3% per chromosome in all generations, the average number of generations needed to find the longest snake is 2,970.8 with a maximum of 14,065 generations. However, if the mutation rate is changed to 100% per chromosome after generation 1,000, the average is 607.5 with a maximum of 1,078 generations. If the mutation rate is changed to 100% per chromosome after generation 500, then the average number of generations necessary to find the longest snake is 322.2 with a maximum of 564 generations. If the mutation rate is changed to 100% per chromosome after generation 250, the average number of generations required is 181.0 with a maximum of 412 generations. The bimodality presented in previous cases is changed to an unimodal distribution when the mutation rate is variable according to the number of generations<sup>9</sup> (see Figure 11).

### 6. Open Questions & Conjecture of Upper Bound

Many questions arise with the problem of finding snakes in hypercubes but the one that has the attention of

<sup>9</sup>When the mutation rate is variable and changed to  $Pop*Generation$  or Equation 8 the number of generations is less than 100, so there is no distribution according to the range selected.

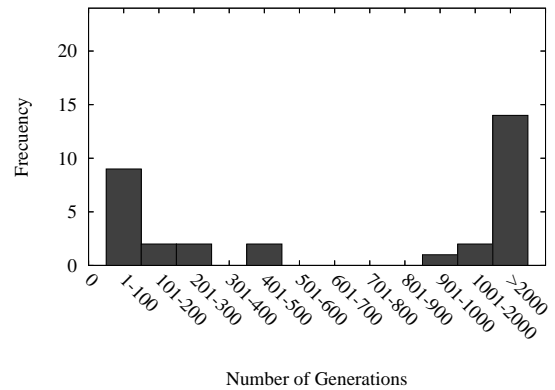


Fig. 8. Histogram for longest snakes using Equation 3 Generations until global maximum reached. Mutation rate 3% per chromosome fixed in all generations.

many researchers is that the length of maximal snakes and coils is unknown. Various conjectures have been proposed for the lower and upper bounds on the length of longest snakes and coils [1], [5], [7], [12], [21], [22]. This article conjectures an upper bound on the number of points of longest snakes as being

$$\#P \leq 1 + 2^{d-1} - \sum_{i=1}^{d-3} i, \text{ for } d \leq 7 \quad (9)$$

and

$$\#P \leq 1 + 2^{d-1} - \sum_{i=1}^{d-2} i, \text{ for } d > 7. \quad (10)$$

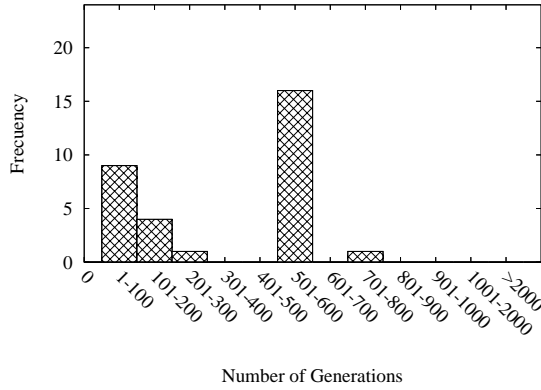


Fig. 9. Histogram for longest snakes using Equation 3. Generations until global maximum reached. Mutation rate 3% per chromosome when generation < 500 and 100% per chromosome when generation ≥ 500.

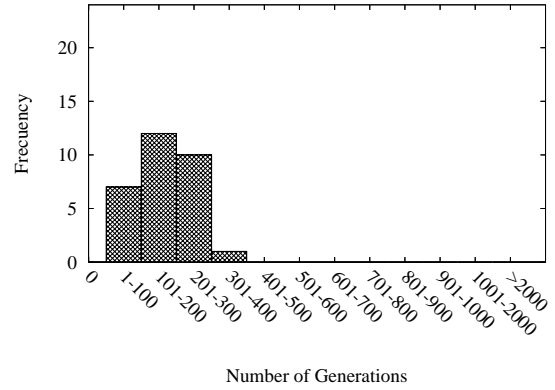


Fig. 11. Histogram for longest snakes Equation 3. Generations until global maximum reached. Mutation rate variable according to the generation.

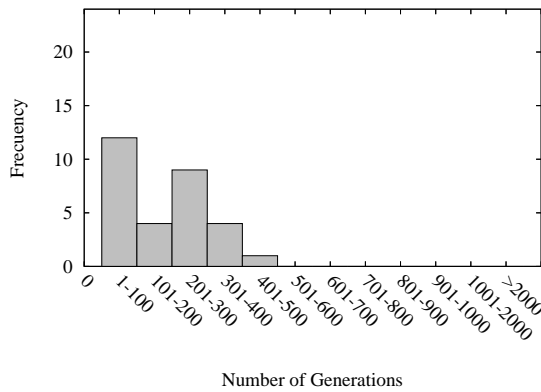


Fig. 10. Histogram for longest snakes using Equation 3 Generations until global maximum reached. Mutation rate 3% per chromosome when generation < 250 and 100% per chromosome when generation ≥ 250.

This conjecture is related to the conjecture of the number of points that a longest snake can traverse in the next dimensional hypercube. For example, if a  $(d - 1)_l$ -dimensional hypercube<sup>10</sup> has a longest snake of length  $n$ , then the maximum number of points that can be used in the next  $(d - 1)_u$ -hypercube is  $(2^{d-1} - n)$ , because  $2^{d-1}$  is the number of new nodes given by the  $(d - 1)_u$ -hypercube and  $n$  cannot be used because those were already used in the  $(d - 1)_l$ -hypercube. The question is if  $n$  nodes of the  $(d - 1)_u$ -hypercube have been invalidated by the longest snake in the  $(d - 1)_l$ -hypercube, how many new nodes are going to be invalidated in the new  $(d - 1)_u$ -hypercube as the snake is growing? Suppose those are going to be  $m$  nodes, then the number of points of the snake in the  $d$ -hypercube would be  $\#P = (n + 1) + 2^{d-1} - n - m = 2^{d-1} - m + 1$ . So  $m$  is the number of new nodes that are invalidated by the snake in the  $(d - 1)_u$ -hypercube and that corresponds to the  $\sum i$  term in Equations 9 and 10. In general, if a point  $p$

<sup>10</sup>A  $d$ -dimensional hypercube can be seen as two  $d - 1$ -dimensional sub-hypercubes:  $(d - 1)_l$ -hypercube and  $(d - 1)_u$ -hypercube

$d$	Conjecture	Klee	Snevily	Casella
3	5	-	-	5
4	8	-	-	8
5	14	-	14	14
6	27	32.0	26	27
7	55	63.9	50	51
8	101	127.9	98	98
9	221	255.9	194	187
10	468	511.8	386	359
11	970	1,023.7	770	681
12	1,983	2,047.6	1,540	1261

Table XIV. Upper bounds on the number of points in longest snakes. Klee [12] and Snevily [21] correspond to upper bounds for coils. Casella [2] corresponds to empirical findings for snakes.

is chosen in the snake path, that point has  $d$  neighbors, one of which can be chosen as the next link in the path—if  $p$  is not the last one and if the next point  $p + 1$  does not violate the constraint—then  $p$  can invalidate  $(d - 1)$  nodes if it is a head or a tail or at most  $(d - 2)$  otherwise. Table XIV compares theoretical results [22] with coils<sup>11</sup> and practical results [2] for dimension  $d \leq 12$ .

One question that arises now is, given the length of a snake in a  $d$ -dimensional hypercube, how many different snakes does the hypercube contain of that length—excluding possible isomorphisms? The reason for this question is that, depending on the number of snakes (solutions), finding one can be more or less difficult<sup>12</sup>.

Another question can be formulated as, when is a longest coil minus one point a longest snake in the  $d$ -dimensional hypercube?<sup>13</sup>

The next interesting topic related to snakes and coils is how, by doing rotations and maybe translations, a snake

<sup>11</sup>For comparison purposes and because of the lack of theoretical findings for bounds for snakes (open paths).

<sup>12</sup>One possible answer could be 0 additional snakes.

<sup>13</sup>Coils are also of practical interest like snakes.

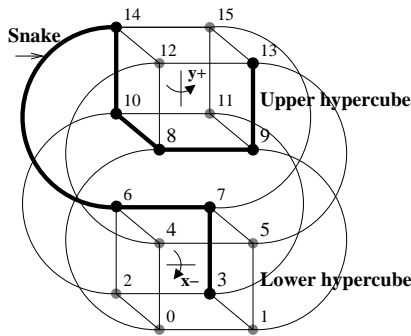


Fig. 12. Snake in a 4-dimensional hypercube to be converted to a coil.

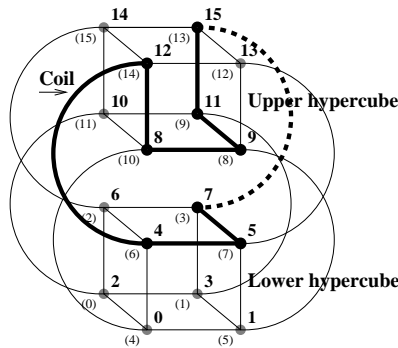


Fig. 13. Snake in Figure 12 converted to a coil doing rotations.

can be converted to a coil. For example, see Figure 12, which is a snake. The lower 3-dimensional hypercube is rotated  $-90^\circ$  around the  $x$  axis, and the upper 3-dimensional hypercube is rotated  $+90^\circ$  around the  $y$  axis to obtain Figure 13. It should be taken into account that the rotation—and possible translation—is done but the original numbering is maintained. For instance, in the previous example the coil is  $7 - 5 - 4 - 12 - 8 - 9 - 11 - 15 - 7$  as is shown in Figure 13.

### 7. Conclusions & Future Work

Different fitness functions for solving the problem of hunting snakes in hypercubes are analyzed. Each function has its strengths and drawbacks that are addressed. The objective and the constraints are the two sides of the coin that are quite difficult to handle in this particular problem because we try to propose fitness functions with “natural” parameters on them. That is, our fitness functions try to use the information of the problem (the objective and constraints) or information that can be derived from the problem (e.g., the number of neighbors of a point) to parametrize the fitness functions. The performance of a genetic algorithm to find an approximate solution is systemic in the sense that it depends not only on the fitness function but also in the initial population, the number of individuals in the initial population, the crossover and

mutation probabilities, the number of generations, and so forth. This article addresses the performance of the genetic algorithm in counting the number of generations to find an optimum in a 4-dimensional hypercube, using the same population size but different initial populations randomly generated, as well as an empirical study of the mutation rate. It is quite difficult to decide which genetic algorithm parameter is more important. A strong parameter is the fitness function in the sense that it is the one that is used in performing selection in looking for “good” solutions. This was one of the principal reasons for devoting some detail to the performance of 7 fitness functions suggested for solving this particular problem. But, as stated in Section 5, the mutation rate is a parameter that can play an important role in finding longest snakes.

This article expands on previous work [4], [5] by analyzing the initial population of the genetic algorithm, the mutation operator, and the way the fitness function evaluates the length of snakes, and conjectures an upper bound on the number of points in longest snakes in hypercubes. Future work includes scaling to higher dimensional hypercubes, looking at different parameters, and looking at parameters in combination.

### References

- [1] H. Abbott and M. Katchalski, “On the Snake in the Box Problem,” *Journal of Combinatorial Theory, Series B*, vol. 45, pp. 13–24, 1988.
- [2] D. A. Casella and W. D. Potter, “New Lower Bounds for the Snake-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Snakes,” in *Proceedings of the Florida Artificial Intelligence Research Society Conference*, 2005, pp. 264–268.
- [3] C. A. Coello, “A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques,” *Knowledge and Information Systems*, vol. 1, no. 3, pp. 269–308, 1998.
- [4] P. A. Diaz-Gomez and D. F. Hougén, “Genetic Algorithms for Hunting Snakes in Hypercubes: Fitness Function Analysis and Open Questions,” in *Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2006, pp. 389–394.
- [5] —, “The Snake in the Box Problem: Mathematical Conjecture and a Genetic Algorithm Approach,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006, pp. 1409–1410.
- [6] —, “A Trade-Off of the Schema Theorem,” in *Proceedings of the 2007 Conference on Artificial Intelligence and Pattern Recognition*, 2007, to appear.

[7] P. G. Emelyanov and A. Lukito, "On the Maximal Length of a Snake in Hypercubes of Small Dimension," in *Discrete Mathematics*, 2000, pp. 51–59.

[8] D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic Algorithms, Noise, and the Sizing of Populations," in *IlligAL Report No. 91010*, 1991, Illinois Genetic Algorithms Laboratory.

[9] D. S. Greenberg and S. N. Bhatt, "Routing Multiple Paths in Hypercubes," in *Proceedings of the Annual ACM Symposium on Parallel Algorithms and Architectures*, 1990, pp. 45–54.

[10] J. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[11] Z. M. Jaroslaw Arabas and J. Mulawka, "GAVaPS—a Genetic Algorithm with Varying Population Size," in *1995 IEEE International Conference on Evolutionary Computation*, vol. 1, 1995, pp. 73–78.

[12] V. Klee, "What is the Maximum Length of a d-Dimensional Snake," *American Mathematics Monthly*, vol. 77, no. 1, pp. 63–65, January 1970.

[13] V. K. Koumoussis and C. P. Katsaras, "A Saw-Tooth Genetic Algorithm Combining the Effects of Variable Population Size and Reinitialization to Enhance Performance," in *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, 2006, pp. 19–28.

[14] S. Lakshmivarahan and S. Dhall, *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems*. MacGraw-Hill, 1990.

[15] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[16] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Bayesian Optimization Algorithm, Population Sizing, and Time to Convergence," in *IlligAL Report No. 2000001*, 2000, Illinois Genetic Algorithms Laboratory.

[17] A. Piszcz and T. Soule, "Genetic Programming: Analysis of Optimal Mutation Rates in a Problem with Varying Difficulty," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006, pp. 953–954.

[18] —, "Genetic Programming: Optimal Population Sizes for Varying Complexity Problems," in *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, 2006, p. 906.

[19] W. D. Potter, R. W. Robinson, J. A. Miller, K. Kochut, and D. Z. Redys, "Using the Genetic Algorithm to Find Snake-In-The-Box Codes," in *Proceedings of the International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, 1994, pp. 421–426.

[20] D. S. Rajan and A. M. Shende, "Maximal and

Reversible Snakes in Hypercubes," in *Proceedings of the Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computing*, 1999.

[21] H. S. Snevily, "The Snake-in-the-Box Problem: A New Upper Bound," in *Discrete Mathematics 133*, 1994, pp. 307–314.

[22] E. W. Weisstein, "Snake," 2007, From MathWorld – A Wolfram Web Resource. Accessed February 2007. [Online]. Available: <http://mathworld.wolfram.com/Snake.html>



**Pedro A. Diaz-Gomez** is a PhD candidate and Master of Science in computer science at the University of Oklahoma, USA. He has a Bachelor's degree in Systems and Computer Engineering from Los Andes University in Colombia, where he also pursued post-graduate work in Telematique and Human Resources. He is Specialist in Total Quality Control and Productivity from El Valle University. He has a second Bachelor's degree in Pedagogic studies, with emphasis in Mathematics, from the Universidad Pedagogica Nacional in Colombia. Mr. Diaz-Gomez's research interest is Genetic Algorithms, which he has applied in computer security and to the snake-in-the-box problem. He is also a Research Assistant at the Robotics, Evolution, Adaptation, and Learning Laboratory and Teaching Assistant at the University of Oklahoma. He has given presentations at international Conferences in the United States, Colombia, and Mexico. He has experience as a Computer Security Consultant, Control and Politics Manager, Production and Computer Center Manager, besides having broad experience as a Computer Science Department Director and teacher. He is also an ACM and AAAI member.



**Dean F. Hougen** is an Assistant Professor in the School of Computer Science at the University of Oklahoma. Dr. Hougen received his BS in Computer Science from Iowa State University in 1988, with minors in Philosophy and Mathematics, and his PhD in Computer Science from the University of Minnesota in 1998, with a graduate minor in Cognitive Science. With over 60 refereed publications in many areas of artificial intelligence, his primary research involves robotics and machine learning, focusing

on distributed heterogeneous robotic systems and situated learning in real robotic systems, including reinforcement learning, connectionist learning, and evolutionary computation. He has also worked in the areas of computer security, expert systems, decision support systems, geographic information systems, and user interfaces, developing several fielded systems in these areas. At the University of Oklahoma he founded and directs the Robotics, Evolution, Adaptation, and Learning Laboratory (REAL Lab) which is his primary research laboratory; the Robotic Intelligence and Machine Learning Laboratory (RIML Lab) which is an interdisciplinary teaching laboratory utilized by the Schools of Computer Science, Electrical & Computer Engineering, and Aerospace & Mechanical Engineering; and the Artificial Intelligence Research Laboratories (AIR Labs) which is an alliance of more than a half-dozen research and teaching laboratories spanning several disciplines. Before coming to Oklahoma, he was an Assistant Professor in the Department of Computer Science and Engineering and Associate Director of the Center for Distributed Robotics, both at the University of Minnesota.