Genetic Algorithms for the Development of Real-Time Multi-Heuristic Search Strategies

Man-Tak Shing and Gary B. Parker*
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Abstract

Search of an unknown space by a physical agent (such as an autonomous vehicle) is unique in search. There is a real-time aspect since the agent is actually moving; using energy each step of the way. The customarily most important goal (to reduce the computation time required to obtain the shortest distance) is not as important as minimal movement. Having limited energy resources and knowledge of the terrain (only adjacent nodes), the key factor for the physical agent's search algorithm is reduction of steps. Any heuristic that can help keep step count to a minimum must be considered. In this paper, we present a simple genetic-algorithm-based method to produce adaptive, efficient multi-heuristic search strategies for the real-time problem. Extensive empirical study shows that this approach produced search strategies with much better performance than existing search algorithms for most terrain types. The methodologies used to develop these improved strategies for our specific case, are also applicable to a multitude of real-time search/optimization problems in the general case.

1 INTRODUCTION

Search of an unknown space by a physical agent (such as an autonomous vehicle) is unique in search. There is a real-time aspect since the agent is actually moving; having limited time to determine its next move and using energy each step of the way. The physical agent traversing a terrain in the real-time problem knows only its current position, the goal's position, and whether adjacent and previously adjacent nodes are passable or not. It learns about the terrain only as it moves from node to node examining all nodes adjacent. Information about past nodes, visited or adjacent, can be stored to build up its knowledge base. Computational time to determine the next move is important, as stopping to compute before each move is undesirable. On the other hand, insufficient computations can result in unnecessary steps and wasted energy.

Having limited energy resources and knowledge of the terrain the key factor in the physical agent's search is the reduction of physical steps. Papadimitriou and Yannakakis (1989) showed that the computational problem of deriving optimal search strategies for the real-time problem is PSPACE-complete. Hence, any heuristic that can help keep step count to a minimum must be considered. Korf (1990) studied this problem and developed the *real-time-A* search*, which uses, for every adjacent node v, the physical agent's distance from the node ($g(v)$) in addition to the distance from goal heuristic ($h(v)$) to determine the best next move by minimizing the objective function $f(v) = g(v) + h(v)$. Shing and Mayer (1991) developed *persistence search* which included a persistence factor (pf = 0.0 to 1.0) to bias the distance from current node. The next move is determined by minimizing the objective function $f(v) = pf \times g(v) + h(v)$ for every frontier node v. Experimental results led to the conclusion that the pf factor could be adjusted to optimize search depending on terrain type and the density of obstacles.

Extending on these works, we believe a combination of additional heuristics can be beneficial in minimizing physical agent steps. As the number of heuristics increases, it is essential to have some efficient means of assigning bias factors to various heuristics to produce an effective multi-heuristic search for different terrain types and densities of obstacles. If the combinatorial explosion required to produce all possible combinations of heuristics is not intractable, the required testing of each to select the best makes this means computational-

---

* LCDR Gary B. Parker, USN is stationed on the USS AMERICA. Research reported here was done while LCDR Parker was with the Computer Science Department, Naval Postgraduate School.

ly prohibitive. Since enumeration is probably not possible, some random means of attaining a near-optimal combination seems to be the most plausible. DeJong (1975) made clear the advantages of genetic algorithms over purely random selection, and Grefenstette *et al* have successfully used SAMUEL, a learning system based on genetic algorithm, to assist autonomous agents to learn robust reactive strategies in evasion, tracking, mine avoidance and local navigation problems (Grefenstette *at al* 1988, Grefenstette 1991, Schultz 1991).

In this paper, we present a simple genetic-algorithm-based method to produce adaptive, efficient and effective multi-heuristic search strategies for the real-time problem. The genetic-algorithm-based learning reported in this paper is "off-line" learning, as oppose to the "anytime" learning conducted by SAMUEL (Grefenstette 1992). Here, the robot cannot change its search strategies in the middle of its maze exploration. Extensive empirical study shows that a genetic algorithm, even with only very simple crossover and mutation operators, can produce search strategies with much better performance (reduced number of steps without prohibitive computation time) than existing search algorithms for most terrain types. The methodologies used to develop these improved strategies for our specific case are also applicable to a multitude of real-time search/optimization problems in the general case.

2 PROBLEM MODEL

To best demonstrate the effectiveness of the multi-heuristic search strategies produced by a genetic algorithm, we chose to apply the strategies to random obstacle distributions in the form of a two-dimensional 64×64 grid of squares (nodes). Nodes can be either free or obstacles, movement can be in eight directions through free spaces only. A perimeter surrounding this grid is a solid row/column of obstacles. The distance from a node to its horizontal/vertical neighbor is 1.0; to its diagonal neighbor is $\sqrt{2}$. The total distance traveled from start to goal according to any search scheme is the sum of each of these individual steps. The effectiveness (fitness) of a specific search scheme is the ratio of the shortest path length from start to goal divided by the distance traveled. Given as a percentage, 100 is the best possible; meaning the distance traveled is equivalent to the shortest path. Specific nodes of the grid can be identified by Cartesian coordinates with the left border column being the y axis and the bottom border row being the x axis. The lowest left node is (1,1); the top right is (64,64).

The grid is internally represented as a 66×66 two dimensional array (the perimeter nodes are all marked as obstacles) made up of pointers to node records. The records store information pertinent to terrain, search (heuristics), graphic display, and pointers to other node records (used in the program for various dynamic structures). The heuristic values stored include *distance_from_start*, *distance_from_goal*, *distance_from_current*, *side_congestion*, *diagonal_congestion* and *subtotal* (refer to Section 3.1 for details). No other node records are used in the program; other structures requiring nodes are set up using pointers to these records.

The 64×64 search space grid is divided into 16×16 density blocks, each containing 4×4 nodes and having a specified block density. Block densities range from 0-15. A block density of 9 means that, on average, nine of the block's 16 nodes will be obstacles (chosen at random). These density blocks are numbered from (0,0) to (15,15) where (0,0) is the bottom left and (15,15) is the top right. Start and goal positions are specified by density blocks. Most of the empirical studies reported in this paper chose the start and goal node from blocks (2,2) and (13,13) respectively. The specific start/goal node is located randomly in that block.

3 MULTI-HEURISTIC SEARCH

The input to the Multi-heuristic Search algorithm consists of a **start** and a **goal** location in an unknown maze, and a set of heuristics and their corresponding bias factors. The set of heuristics is partitioned into two groups, *stable heuristics* and *unstable heuristics* (see Sections 3.1 and 3.2 for details). The algorithm works as follows:

```
current := start;
while current ≠ goal loop
   for all nodes v within 2 moves of current loop
      if v is adjacent and untouched then
         v.subtotal := v.stable_heuristics_vector •
                                    respective_biases_vector;
         add v to frontier_heap;
            /* the top of frontier_heap contains */
            /* the node with minimum subtotal   */
      else if v is frontier and any stable_heuristics
                                     of v have changed then
         v.subtotal := v.subtotal + adjustment;
         update v's position in frontier_heap;
   end for loop;

   if empty(frontier_heap) then
      return big_number;  /* no solution */
   else
      find frontier node v that minimizes f(v) =
         v.subtotal + v.unstable_heuristics_vector •
                                    respective_biases_vector;
      v.dist_traveled := current.dist_traveled + g(v);
         /* g(v) is the shortest distance through       */
         /* known paths from current to frontier node*/
      current := v and remove v from frontier_heap;
end while loop;
return goal.dist_traveled;
```

3.1 STABLE HEURISTICS

Stable_heuristics have values that will not change when applied to locations more than two steps away from the current node. They include *distance_from_goal* (hg), *distance_from_start* (hs), *side_congestion* (hsc), *diagonal_congestion* (hdc), and *momentum* (hm). The subtotal $\overline{fs}(v)$ is calculated using these functions multiplied by their respective bias factor and stored in *v.subtotal*.

$$fs(v) = hgf \times hg(v) + hsf \times hs(v) + hscf \times hsc(v) + hdcf \times hdc(v) + hmf \times hm(v) \quad \text{(Eq. 3.1)}$$

Distance_from_goal ($hg(v)$) - The Euclidean distance from the node v to the **goal** node. This heuristic is usually considered important in any search. It is used in combination with *distance_from_current* for Persistence Search, and by itself for Best-first Search.

Distance_from_start ($hs(v)$) - This is usually the actual shortest path from the start node to the considered frontier. Currently believed to be useless in a real-time environment, it should be selectively eliminated by natural selection as the genetic algorithm trains. For our implementation, it is approximated by computing the Euclidean distance from start to frontier. It may be significant in some of the more complex terrains that require a switch back.

Congestion - The congestion parameters, *side_congestion* ($hsc(v)$) and *diagonal_congestion* ($hdc(v)$), are attempts to assist the physical object in avoiding areas of increased obstacle density. This reduces exploration of paths through high density areas, favoring the safer path of increased options available in the open space. The parameters are separated in case one is more appropriate than the other. Both would be much more effective if the physical object's perception were not limited to adjacent nodes. If all nodes adjacent to the frontier node could be seen, these factors' importance would increase significantly. The *side_congestion* heuristic examines the known horizontal/vertical neighbors of the frontier node to count the number of obstacles. Nodes with more known obstacle neighbors are less desirable. The minimum value is 0 and 4 is the maximum. The *diagonal_congestion* heuristic is similar to *side_congestion* with the count being made of the frontier node's diagonal vice horizontal/vertical neighbors.

Momentum ($hm(v)$) - This heuristic attempts to avoid zigzag by making forward (in relation to last move) nodes the most desirable. It should be useful in valley/ridge terrains where the best path is straight through the valley. By maintaining momentum, the physical object avoids steps wasted in popping in and out of each crevice which has nodes closer to the goal. Straight ahead movement results in a value of 0, a 45^{o} shift makes it 1, a 90^{o} shift is 2, and a 135^{o} shift or non-adjacent move results in a value of 3 (making only the adjacent nodes subject to change after a move).

3.2 UNSTABLE HEURISTICS

Unstable_heuristics have values that are liable to change as the current node changes. Examples in our case: *distance_from_current* ($hc(v)$) and *move_away_factor* ($hma(v)$). The algorithm minimizes Equation 3.2 below using the efficient "branch-and-bound" search through known (visited) nodes described in Section 4.3 of the paper by Shing and Mayer (1991).

$$f(v) = fs(v) + hcf \times hc(v) + hmaf \times hma(v) \quad \text{(Eq. 3.2)}$$

Distance_from_current ($hc(v)$) - The distance from the current node to the frontier node; important in Real-Time-A* and Persistence Search to determine if backtracking is worth the steps required. It is the actual distance computed as the actual steps required to move from the current node to the frontier.

Move_away_factor ($hma(v)$) - It attempts to continually reduce the search space by reducing desirability of nodes that increase the x and/or y difference between the current and goal nodes. Increasing either the x or y distance counts as 2, increasing both counts as 4, and no increase results in the heuristic having a value of 0.

3.3 CHROMOSOME REPRESENTATION

A 32 member array of individual records makes up the population. Each stores the individual's fitness and its chromosome which contains biases for the search heuristics. The chromosome is a 32 bit unsigned integer; subdivided into eight four-bit alleles [A1 A2 A3 A4 A5 A6 A7 A8]. Each allele represents a bias factor with a range from 0 to 15.

The *hsf, hgf, hcf, hscf, hdcf, hmaf* and *hmf* bias factors are stored in the individual chromosome's lower 28 bits, i.e. A2 through A8. The values of these bias factors are set during training. The four bits in A1 are, in our implementation, a place holder for future additional heuristics since only seven applicable heuristics were identified.

3.4 GENETIC ALGORITHM

The task of the genetic algorithm is to find the combination of the seven bias factors that will result in the optimum search scheme stored in a single individual's chromosome. Application of genetic operators to a population (32 in our case) of these individuals will, after numerous iterations (1000 generations in our case), produce our desired optimal individual.

During training, the genetic algorithm is invoked once after each predetermined number of cycles (5 in our case) making up one generation. The input population will have a fitness value (ability to get through the terrain) assigned to each of it's 32 individuals (details of this process are described in Section 4.2). Our genetic algorithm makes use of four genetic operators: *selection, allele crossover, bit crossover* and *mutation*. The allele crossover operator generates new strategies (with a 0.86 probability) by cross-mixing individuals at randomly chosen allele boundaries. The bit crossover operator generates new bias value for a randomly chosen allele (with a 0.53 probability) by cross-mixing bits between the corresponding alleles of two individuals. Mutation is conducted on a bit-by-bit basis with a 0.005 probability. Selection is done using a roulette wheel similar to the algorithm presented in chapter one of the text by Goldberg (1989), with the additions of allowing the best two individuals to go unchanged. The result is similar to DeJong's R2 elitist model (DeJong 1975).

4 EVALUATION

To evaluate the effectiveness of the bias factors generated by the genetic algorithm, ten populations were trained and compared to previously established search strategies, using ten different density distributions

4.1 DENSITY DISTRIBUTIONS (TERRAIN TYPES)

Once the different density distributions are established, the block densities remain unchanged from the start of training through testing. Although the block densities remain constant, actual obstacle placement is determined stochastically and changes from run to run. The point is to investigate the adaptability of genetic algorithm to produce the best strategy for directing the search through terrains where the general density distribution is known but actual obstacle placement is not. Ten sets of block densities were used to simulate ten different terrain types. The first six terrain types are considered natural terrains since they closely resemble actual topological conditions. The start density block is always (2,2) unless otherwise stated. The goal density block is always (13,13) unless otherwise stated. See Figure 1 for the block density distributions used.

4.2 TRAINING

Training of the population is analogous to selectively breeding a random group of asexual organisms to obtain superior capability in a specific area. The capability one wishes to optimize is transit from start to goal in the least number of steps. The specific area is a specific terrain layout where one has a general idea about obstacle density distribution, but have no information about the location of specific obstacles.

The first step is to generate a series of specific terrains from a general obstacle density distribution. This can be done by placing obstacles in each area if a randomly generated number is less than the specified density. In our implementation, we simply loop through the 64×64 node array assigning each nodes state to OBSTACLE if the random number is less than the density value of the corresponding density block. The second step is to generate a population of 32 individuals giving them randomly generated chromosomes. Now the training begins:

```
for each of 1000 generations loop
   for each of 5 cycles loop
      loop until a successful A* search;
         create a terrain from the density_array;
         shortest_path := A* search;
      end until loop
      run each individual through the terrain
         accumulating its fitness_sum by comparing
         its actual path to the shortest path;
   end for loop;

   compute each individual's average fitness from
      fitness_sum and number of cycles;
   apply the genetic algorithm to the population;
end for loop;
return a trained population;
```

4.3 TESTING

Testing of the trained populations was performed by comparing the search conducted by the best individual in each population to searches accomplished using Hill-climbing (Winston 1992), Best-first (Winston 1992), Real-Time-A* (Korf 1990), and Persistence Search (Shing and Mayer 1991). The following equation was used to compute fitness for all search schemes:

$$fitness = integer(((shortest\_path)/(actual\_path))\times 100) \quad \text{(Eq. 4.1)}$$

Each search scheme was tested on 500 distinct terrains produced using the corresponding density matrix. Average fitnesses over the 500 were assigned and a comparison of these fitnesses is presented in the results.

Our implementation of the Persistence Search is a modification of the original work, where the equation $f(v) = pf \times g(v) + h(v)$ in the paper by Shing and Mayer (1991) is replaced by Equation 4.2 shown below. The gf and hf factors are introduced to effectively replace/discretize/expand the persistence factor (pf) which can have any value between 0.0 and 1.0. We found that an infinite range of possibilities for this factor was not required. A discrete, yet sufficient, span can be obtained by setting gf and hf to any number of possibilities where $gf \le hf$. Setting hf to 15 and incrementing gf from 0 to 15 gives us the equivalent of a 0.0 to 1.0 range with increments of 0.067 each.

$$f(v) = gf \times g(v) + hf \times h(v) \quad \text{(Eq. 4.2)}$$

There is also now the expanded capability of having the $g(v)$ be the more important factor in the search ($gf > hf$). The best values for the gf and hf bias factors for the modified Persistence Search were determined before testing by running 32 combinations (chromosomes of 00f00000 to 00ff0000 and 000f0000 to 00ff0000) through 50 distinct terrains. From this, the best combinations of the two factors was used to represent Persistence Search. Likewise, the best bias_factor_vector used by the Multi-Heuristic Search for each terrain type was chosen by running the individuals in the populations produced by the genetic algorithm through 50 distinct terrains of the corresponding terrain type. The individual with the highest fitness was chosen to represent the GA-trained population.

4.4 EXPERIMENTAL

The fitness of each search scheme in these results is the number of its required steps divided by the minimum steps possible (Eq. 4.1), averaged over the 500 terrains used for testing in each terrain type. Fitness is presented as a percentage, with a 100% search scheme being one that can, on the average, search a terrain type in the minimum steps possible. In general, the easier the density layout of the terrain, the higher the fitness will be. A graph comparing the fitness of applicable search schemes is presented for each natural terrain density layout (Figure 2). The genetic algorithm was extremely successful in producing the best search strategies for all natural terrains. A closer look at the resultant chromosomal make-ups reviews that the genetic algorithm was able to learn the characteristics of different terrains and produce bias factors to take advantage of the different situations. Although the genetic-algorithm-produced strategies was always at least as good as the next best in all random terrains, it was not a substantial improvement over Persistent Search. Only in the most complex of the four random terrains did the genetic-algorithm-produced scheme really excel. This seems to suggest that the additional heuristics are only essential in natural terrains where some pattern in obstacle density exists or in random terrains of high complexity. As suggested by DeJong (1992), the genetic algorithm can only optimize to a certain point (depending on implementation) before reaching a state of dynamic equilibrium. The first three random terrains were of insufficient complexity to allow the genetic algorithm to convincingly surpass all conventional search schemes.

Since the genetic-algorithm-produced search strategies are substantially better for our natural terrains and as least as good as standard search schemes for random terrains, they should be advantageous to use on any actual natural terrain. This is of course contingent on the physical agent's dependence on minimal steps and its computational speed. If it's computational speed is sufficient to avoid delays before each step and/or minimal steps are essential, the genetic algorithm produced scheme should always be used. Table 1 shows a comparison of the average time required for each strategy to search from start to goal for each of the terrains. As expected, the more complicated strategies require additional computation time, but are not considered slow enough to prohibit their use except in cases of high speed agents with slow computational speed.

5 CONCLUSIONS

Heuristics previously used for search of an unknown space by a physical agent are *distance_from_goal* and *distance_from_current*. These are insufficient to minimize energy expenditure (steps taken) when some general knowledge of the area is known. The additional heuristics found to be pertinent are *distance_from_start*, *congestion* factors which account for obstacle density around the considered frontier node, *move_away_factor* which encourages reduction of the search space, and *momentum* which avoids wasted steps in course variations. These seven heuristics with their proper individual biases were found to be superior to standard search schemes. In this paper, we show that genetic algorithms, even with only very simple crossover and mutation operators, can produce very effective heuristic biases that are adaptable to unknown search spaces if some general knowledge of the search space is available. Training done with randomly generated search spaces having common characteristics lead to robust search schemes which are, on the average, more fit than previously used strategies.

We believe that this methodology of fitting a set of known heuristics into a binary representation, and applying genetics-based training is also applicable to a multitude of real-time search/optimization problems. Tests in other specific areas are needed to prove our conjecture. In addition, further research could be done in the application of more advanced genetic-algorithm-based learning techniques. The approach reported in this paper addresses only the problem of optimizing a set of known heuristics before actual maze exploration. A more challenging problem is to have the robot conduct "real-time" learning, modify its strategy as it learns more about the maze through actual exploration and generate new heuristics on its own. More sophisticated representations of the search space, perhaps similar to those used by SAMUEL (Grefenstette *et al* 1990), are needed to support these complex learning activities.

Acknowledgements

Work on this paper was supported in part with funds provided by the Naval Postgraduate School.

References

K. DeJong (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

K. DeJong (1992). Genetic Algorithms Are NOT Functional Optimizers. Technical Report, Computer Science Department, George Mason University.

J. Grefenstette (1991). Lamarckian Learning in Multi-agent Environments. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 303-310). San Diego, CA.

D. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Ma: Addison-Wesley. 1989.

J. Grefenstette, C. Ramsey and A. Schultz (1990). Learning Sequential Decision Rules Using Simulation Models and Competition. *Machine Learning 5(4)* (pp. 355-381).

J. Grefenstette and C. Ramsey (1992). An Approach to Anytime Learning. *Proceedings of the Ninth Machine Learning Conference*. Aberdeen, Scotland.

P. Hart, N. Nilsson and B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern. SSC-4(2)* (pp. 100-107).

R. Korf (1990). Real-Time Heuristic Search. *Artificial Intelligence 42(2-3)* (pp. 189-211).

C. Papadimitriou and M. Yannakakis (1989). Shortest Paths Without a Map. *Proceedings of the 1989 ICALP Conference.*

A. Schultz (1991). Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation. *Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology* (pp. 213-225). Durham, NH.

M. Shing and M. Mayer (1991). Persistence Search - A New Search Strategy for the Dynamic Shortest Path Problem. Technical report NPSCS-91-011, Computer Science Dept., Naval Postgraduate School.

P. Winston (1992). *Artificial Intelligence*. Reading, Ma: Addison-Wesley.

Table 1: Average Seconds Required To Search Each Terrain

	Best First	Persistence	Hill Climb	Real-Time-A*	GA-produced
central mountain	0.0239	0.0274	0.0065	0.0163	0.0615
single left ridge	0.0232	0.0264	0.0075	0.0176	0.0694
single right ridge	1.3051	0.1496	0.0199	0.1405	0.2718
double ridge	3.2553	1.3562	0.0420	0.4131	2.3420
single left plateau	0.1071	0.0982	0.0167	0.0626	0.1481
plateau with ridge	0.0769	0.0856	0.0146	0.0487	0.1310
random one	0.0177	0.0265	0.0064	0.0114	0.0478
random two	0.0313	0.0267	0.0056	0.0150	0.0644
random three	0.0294	0.0258	0.0065	0.0151	0.0521
random four	0.1629	0.2255	0.0165	0.0481	0.1563

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1
1	3	5	5	5	5	5	5	5	5	5	5	5	5	3	1
1	3	5	7	7	7	7	7	7	7	7	7	7	5	3	1
1	3	5	7	9	9	9	9	9	9	9	9	7	5	3	1
1	3	5	7	9	b	b	b	b	b	b	9	7	5	3	1
1	3	5	7	9	b	d	d	d	d	b	9	7	5	3	1
1	3	5	7	9	b	d	f	f	d	b	9	7	5	3	1
1	3	5	7	9	b	d	f	f	d	b	9	7	5	3	1
1	3	5	7	9	b	d	d	d	d	b	9	7	5	3	1
1	3	5	7	9	b	b	b	b	b	b	9	7	5	3	1
1	3	5	7	9	9	9	9	9	9	9	9	7	5	3	1
1	3	5	7	7	7	7	7	7	7	7	7	7	5	3	1
1	3	5	5	5	5	5	5	5	5	5	5	5	5	3	1
1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Central Mountain

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
4	4	4	4	4	4	4	4	4	4	4	2	2	2	2	2
6	6	6	6	6	6	6	6	6	6	6	4	2	2	2	2
8	8	8	8	8	8	8	8	8	8	8	6	4	2	2	2
a	a	a	a	a	a	a	a	a	a	a	8	6	4	2	2
c	c	c	c	c	c	c	c	c	c	c	a	8	6	4	2
f	f	f	f	f	f	f	f	f	f	f	c	a	8	6	4
f	f	f	f	f	f	f	f	f	f	f	c	a	8	6	4
c	c	c	c	c	c	c	c	c	c	c	a	8	6	4	2
a	a	a	a	a	a	a	a	a	a	a	8	6	4	2	2
8	8	8	8	8	8	8	8	8	8	8	6	4	2	2	2
6	6	6	6	6	6	6	6	6	6	6	4	2	2	2	2
4	4	4	4	4	4	4	4	4	4	4	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Single Left Ridge

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4
2	2	2	2	4	6	6	6	6	6	6	6	6	6	6	6
2	2	2	4	6	8	8	8	8	8	8	8	8	8	8	8
2	2	4	6	8	a	a	a	a	a	a	a	a	a	a	a
2	4	6	8	a	c	c	c	c	c	c	c	c	c	c	c
4	6	8	a	c	f	f	f	f	f	f	f	f	f	f	f
4	6	8	a	c	f	f	f	f	f	f	f	f	f	f	f
2	4	6	8	a	c	c	c	c	c	c	c	c	c	c	c
2	2	4	6	8	a	a	a	a	a	a	a	a	a	a	a
2	2	2	4	6	8	8	8	8	8	8	8	8	8	8	8
2	2	2	2	4	6	6	6	6	6	6	6	6	6	6	6
2	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Single Right Ridge

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	4	6	6	6	6	6	6	6	6	6	6	6
2	2	4	6	8	a	a	a	a	a	a	a	a	a	a	a
6	6	8	a	c	f	f	f	f	f	f	f	f	f	f	f
4	6	6	8	a	c	c	c	c	c	c	c	c	c	c	c
4	4	6	6	8	a	a	a	a	a	a	a	a	a	a	a
4	4	4	6	6	8	8	8	8	8	8	8	8	8	8	8
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
8	8	8	8	8	8	8	8	8	8	8	6	6	4	4	4
a	a	a	a	a	a	a	a	a	a	a	8	6	6	4	4
c	c	c	c	c	c	c	c	c	c	c	a	8	6	6	4
f	f	f	f	f	f	f	f	f	f	f	c	a	8	6	6
a	a	a	a	a	a	a	a	a	a	a	8	6	4	2	2
6	6	6	6	6	6	6	6	6	6	6	4	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Double Ridge

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Single Left Plateau

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
c	c	c	c	c	c	c	c	c	c	c	c	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	c	a	8	2
c	c	c	c	c	c	c	c	c	c	c	c	c	a	8	2
a	a	a	a	a	a	a	a	a	a	a	a	a	a	8	2
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Single Left Plateau with Ridges

d	2	b	e	f	1	6	b	f	e	b	b	1	0	f	f
b	a	8	6	8	1	2	4	f	9	9	b	6	b	c	3
d	8	2	d	9	8	8	8	6	3	3	7	3	2	7	f
c	f	5	4	0	8	8	f	1	2	b	7	d	7	b	b
f	d	8	8	5	0	1	c	4	4	3	7	7	a	6	3
9	c	8	a	4	0	9	5	2	4	d	0	c	8	b	b
5	3	4	a	3	5	6	7	9	a	f	0	4	5	4	e
1	c	8	5	c	1	b	f	6	8	f	2	0	a	d	5
d	1	f	0	6	6	8	0	0	7	0	4	c	4	2	e
0	a	3	d	c	e	c	2	6	b	4	6	5	1	b	2
3	b	2	9	1	a	9	1	1	a	5	e	e	8	c	f
2	f	c	e	e	8	0	4	3	4	b	8	6	6	a	9
1	c	2	2	7	c	3	8	6	9	6	4	1	2	3	3
2	f	2	0	7	2	4	a	7	f	2	d	6	c	6	7
9	8	a	0	4	d	8	a	6	f	f	7	1	2	b	3
2	d	3	9	f	8	4	6	7	6	3	d	3	9	5	c

Random One

6	4	6	c	d	c	8	9	a	d	a	c	4	6	e	5
4	f	7	b	5	9	0	4	5	5	a	2	8	d	9	e
1	0	b	f	c	3	8	6	0	2	2	4	8	0	a	d
f	1	8	4	a	9	8	f	e	3	2	6	0	b	5	2
b	0	1	7	3	9	e	4	c	0	8	4	1	2	1	0
3	a	5	e	3	d	d	1	0	f	8	1	b	d	3	6
d	4	e	0	d	c	4	9	c	d	e	d	f	f	e	3
9	3	1	c	0	e	e	1	e	6	2	9	3	5	f	0
9	d	0	6	9	5	0	6	2	e	3	1	d	1	4	7
4	5	3	5	4	1	6	2	7	8	b	a	d	a	a	6
8	b	c	1	0	c	7	2	a	b	3	8	c	8	f	1
d	2	6	1	4	c	3	b	4	e	6	1	9	0	7	1
b	3	2	b	0	a	d	a	5	1	2	1	9	1	2	6
4	8	8	8	4	b	3	8	a	9	9	3	a	0	4	5
4	6	1	4	0	e	e	5	f	1	7	8	2	9	f	6
2	7	e	6	2	2	f	c	b	8	f	5	9	3	b	d

Random Two

2	c	a	4	5	0	0	1	e	1	e	8	8	e	2	d
0	b	b	7	1	9	8	4	d	5	6	6	0	5	5	2
1	0	7	7	0	7	8	e	8	6	6	0	4	8	e	5
3	9	c	4	2	5	8	f	a	f	6	a	4	b	d	6
b	4	d	b	b	5	a	4	c	0	4	0	9	2	5	c
b	2	1	e	7	9	d	1	8	3	c	d	f	9	3	a
d	0	6	8	5	0	c	1	0	1	2	9	3	7	6	f
9	7	d	0	0	a	2	9	e	e	6	d	7	9	7	4
9	d	c	e	d	9	0	e	a	2	7	d	9	d	c	3
4	9	3	5	4	5	e	2	3	4	f	a	d	6	e	6
4	b	4	1	4	4	f	e	6	7	b	0	4	8	3	9
1	6	e	5	c	c	7	f	0	6	a	d	d	8	3	1
3	7	2	7	c	2	5	2	9	1	2	d	9	5	6	a
c	4	0	8	0	7	7	0	e	1	d	b	a	0	c	d
8	e	5	4	0	a	6	9	b	9	7	4	e	d	f	a
2	f	2	2	6	a	3	4	b	0	f	5	1	b	3	9

Random Three

b	e	5	2	3	b	2	f	9	0	d	9	b	4	1	3
9	0	a	c	e	1	6	4	3	1	7	d	2	7	2	d
5	8	0	9	3	2	8	c	2	5	5	d	9	6	1	3
6	b	f	4	c	6	8	f	7	0	d	9	7	f	7	d
7	7	6	a	9	e	7	c	4	c	9	d	3	a	0	9
5	0	e	2	6	6	1	d	6	e	7	e	e	e	b	5
5	1	0	e	f	7	a	3	3	4	1	6	e	1	0	4
1	e	6	7	4	7	5	b	6	c	9	4	a	4	9	f
5	9	d	4	0	8	8	4	c	9	a	a	a	a	e	c
8	4	3	d	c	8	8	2	4	1	6	e	5	f	d	a
9	b	e	9	3	6	d	f	f	8	9	a	2	8	6	b
c	9	8	8	2	0	a	6	1	0	5	6	0	2	0	9
d	e	2	0	5	0	f	4	8	9	e	a	1	4	5	d
e	d	6	0	d	0	6	e	1	b	4	1	e	4	a	b
3	c	c	8	c	b	c	4	4	b	f	5	f	4	3	d
2	9	d	f	9	4	e	a	f	2	b	d	7	5	9	a

Random Four

Figure 1: Density Distribution

Figure 2: Experimental Results