Initial Performance Comparisons for the Delta Coding Algorithm

Keith E. Mathias[†] and L. Darrell Whitley[†]

Abstract — **Delta coding is an iterative genetic search strategy that sustains search by periodically re-initializing the population. This helps to avoid premature convergence during genetic search. Delta coding also remaps hyperspace with each iteration in an attempt to locate "easier" search spaces with respect to genetic search. Here, the optimization ability of delta coding is compared against the CHC genetic algorithm and a mutation driven stochastic hill-climbing algorithm on a suite of standard genetic algorithm test functions.**

I. Introduction

During the early stages of the search when population diversity is high, most genetic algorithms make rapid improvement in the solutions that are located. However, the degree of progress toward improved solutions becomes asymptotic as the population exhibits less and less diversity [1, 6]. *Delta coding* [14], an iterative genetic search strategy, sustains search by periodically re-initializing the population, thereby avoiding this phenomenon. Delta coding also produces a new mapping of hyperspace at each iteration. This allows delta coding to explore other versions of hyperspace that may be "easier" with respect to genetic search [7].

The delta coding algorithm begins with the initial run of a genetic algorithm. When the population diversity has been adequately exploited the best solution parameters are saved as the *interim solution.* The genetic algorithm is then restarted with a new random population. The parameter substrings in subsequent runs of the genetic algorithm are decoded such that they represent a distance or *delta value* ($\pm\delta$) away from the interim solution parameters. The delta values are combined with the interim solution so that the resulting parameters are evaluated using the same fitness function. This forms a new hypercube with the interim solution at its origin.

Delta coding also provides a mechanism that reduces and enlarges the size of the hypercube currently being searched. The reduction mechanism allows the algorithm to focus the search in subpartitions of hyperspace that appear promising. The expansion mechanism allows the algorithm to re-explore previously overlooked portions of the search space later in the search. Insights on how these mechanisms perform with respect to hyperplane sampling are explored here as well as issues associated with remapping the search space.

Delta coding has been shown to exhibit superior performance when compared with other genetic algorithms in some environments [14]. However, a more comprehensive comparison of the delta coding algorithm on standard test functions is important for general evaluation. Here, we compare its performance against that of the CHC genetic algorithm [4] and a *mutation driven stochastic hill-climbing algorithm*, using the DeJong test suite [3] and the Rastrigin, Schwefel, and Griewank functions [10].

The test functions used here were designed to be difficult for hill-climbing algorithms. However, the mutation driven stochastic hill-climbing algorithm is able to consistently solve all but two of the test functions when Gray coding is used. But when Gray coding is not employed the stochastic hill-climber is able to consistently solve only one of the test functions. This suggests that Gray coding is able to mask the attributes that make these functions difficult. Therefore, the comparisons were performed for the stochastic hill-climber, delta coding and CHC algorithms without Gray coding.

The results of these performance comparisons demonstrate the effectiveness of the delta coding algorithm in optimization. The delta coding algorithm is able to consistently locate the optimal solution for all of the test functions. The CHC algorithm is only able to consistently solve 6 of the 8 functions and the stochastic hill-climber only solves one function consistently. Additionally, delta coding is able to locate the optimal solution in fewer trials than the other algorithms for all but one test function.

II. The Delta Coding Algorithm

Our delta coding [14] implementation uses GENITOR [13] as the basic engine for genetic search. Delta coding evaluates the problem in a normal fashion on the initial iteration except that the population diversity is monitored. This is done by measuring the Hamming distance between the best and worst strings in the current *persisting* population,[1] where only the least significant bit of each parameter may differ. If the Hamming distance is more than one, search continues. Otherwise, search is suspended and the best solution is saved as the *interim solution.*

[†]Department of Computer Science, Colorado State University; Fort Collins, CO 80523 (mathiask, whitley@cs.colostate.edu)

This research was supported in part by a grant from the Colorado Advanced Software Institute.

[1]The worst string in the population is not considered as part of the persisting population since it is always displaced with the insertion of the new offspring.

A new population is then created randomly and GENITOR is restarted. This is the beginning of the *delta iterations*. Each parameter, when decoded for fitness evaluation during a delta iteration, is applied as a *delta value* ($\pm\delta$) to the interim solution saved from the previous iteration. This method of applying the new strings to the interim solution, in effect, searches a new hypercube with the interim solution at its origin. The process of selecting strings, applying crossover and inserting offspring continues while the population diversity is monitored as previously described. At the end of each delta iteration, except the first, the numerical range for each parameter in the string is adjusted to allow the algorithm to search different subpartitions of hyperspace. This is done by altering the number of bits used to represent each parameter, based on the interim solution found in consecutive iterations.

The cycle of converging to a new interim solution, altering the parameter bit representations and restarting the search is repeated until an acceptable solution is found or until a maximum number of recombinations have been executed. However, the final delta iteration is always allowed to complete, even at the expense of a few additional trials. The delta coding algorithm is outlined in Figure 1.

```
NORMAL GENITOR PHASE:
  While (Hamming Distance Between Best and Worst Strings > 1)
  {
      Apply Recombination
      Evaluate Fitness and Insert Offspring
      IF (Fitness < Threshold) THEN
          HALT
  }

TRANSITION PHASE:
  Save Best Solution in Population as Interim Solution
  Re-initialize Population
  Encode Parameters Using (X − 1) Bits, Use Extra Bit as Sign

DELTA ITERATION PHASE:
  While ((Trials < MAX_TRIALS) && (Fitness > Threshold))
  {
      While (Hamming Dist. Between Best and Worst Strings > 1)
      {
          Apply Recombination
          Decode All Parameter String Values
          Add Decoded Values to Interim Solution Parameters
          Evaluate Fitness and Insert Offspring
      }
      Save Best New Solution as Interim Solution
      Re-initialize Population
      IF (Delta Values == 0) THEN
          IF (Parameter Length < Original Length) THEN
              Encode Parameters Using 1 Additional Bit
      ELSE
          IF (Parameter Length > Lower Bound) THEN
              Encode Parameters Using 1 Less Bit
  }
```

Figure 1: The *Delta Coding* Algorithm.

A. Subpartition Sampling and Delta Parameter Decoding

While the delta coding algorithm is conceptually simple, the delta parameter decoding mechanism can be complex. During the delta iterations, each delta parameter is decoded and applied to its corresponding interim solution parameter by the following method: if the sign bit of the delta value is zero then directly add the delta value to the interim solution parameter; if the sign bit of the delta value is one, complement all of the other bits in the parameter and add the result to the interim solution parameter. All addition is performed mod 2^b, where b is the number of bits in the original encoding for each parameter. Complementing and adding bits to the interim solution is equivalent to subtracting their numeric value from the interim solution.

Application of the delta parameters in this fashion provides a new mapping of the search space for each iteration. Each new mapping places the new interim solution at the origin of the hypercube and forms a new set of hyperplane competitions based on the interim solution and the numerically adjacent values reachable from that interim solution, given the current parameter string lengths.

As described earlier, delta coding alters the length of the string representing each parameter at the end of each delta iteration, except the first. The remapping that occurs immediately after the first iteration of genetic search is unique in that the number of bits used to encode the problem is not altered. Only the mapping is changed, using the first bit in each parameter as a sign bit and the remaining bits as the delta value. This allows the same search space volume to be explored with a new mapping. Alteration of the parameter string length effectively changes the size of the hypercube. This allows the genetic algorithm to search subpartitions within the original search space. The area searched is an interval around the parameters of the interim solution, defined by the reach that is dictated by the parameter string length. The algorithm uses information about consecutive interim solutions to trigger operations that expand and contract the solution space that is currently being explored. This is done by varying the number of bits used to represent each parameter in one of two ways.

First, if the new best solution is different than the previous interim solution, the number of bits used to encode each delta parameter value is reduced by one bit. This effectively shrinks the hypercube. Reductions are limited to some minimal number of bits for the delta parameters to prevent the subpartition currently being searched from becoming too small. The new interim solution need not be more fit than the last, *only different*.

The remapping of the interim solution to the string of all 0's coupled with the application of delta values to the interim solution parameters not only determines how the search space is mapped but also what subpartition of the search space is sampled. This phenomenon is of particular interest when the delta parameter encodings have been reduced so that only part of the search space is sampled in a delta iteration.

Application of delta parameters as delta values to the interim solution guarantees that the points adjacent to the interim solution in numeric space will be contained in the subpartition that is sampled for that delta iteration (Figure 2). There is one exception to this subpartition sampling behavior. If the interim solution is located in close

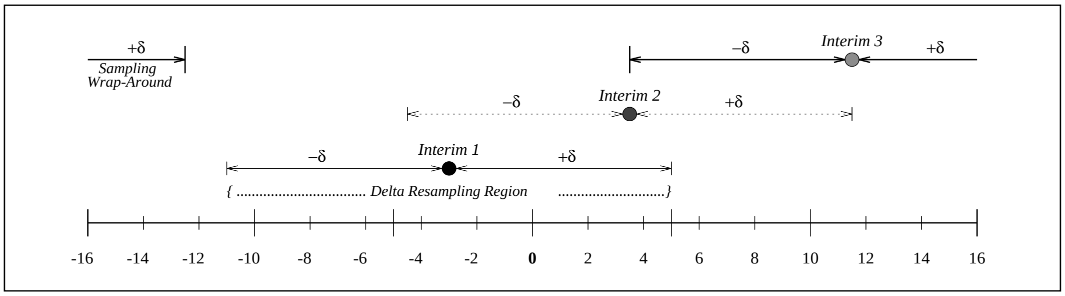

Figure 2: Points Sampled in a 1-dimensional Numeric Space Using Delta Coding.

proximity to any boundary of the search space the sampling will be discontinued at that boundary and resumed on the opposite boundary of the hypercube extending back into the search space (see Interim 3, Figure 2).

The second event causing the number of bits in the parameter encoding to be altered is when a delta iteration converges to exactly the same solution (i.e. delta values of all zeros). When the genetic algorithm converges to the same interim solution, the search has effectively stalled. To prevent this, the number of bits used to encode the delta values ($\pm\delta$) is increased by one bit, expanding the hypercube. This allows the genetic algorithm to search a larger partition of the search space with a mapping that may be different from the one used previously. This would be the case if the current interim solution were different from the interim solution two or more iterations previous. The expansion mechanism is bounded by the initial parameter encoding length.

III. Advantages of Delta Coding

The delta coding algorithm has several advantages. First, it is conceptually simple to understand and implement. Second, it does not rely on disruptive mechanisms to sustain diversity during genetic search. Delta coding searches until the diversity of the current population is exhausted. It then saves the best known solution parameters at that point and re-initializes the population. This provides a new and diverse population with which to resume the search. This resurgence of diversity for each delta iteration allows the search to continue to locate new solutions.

With each new interim solution a new mapping is imposed on the search space. No attempt is made to preserve previous relationships in Hamming space. Exploring new mappings may be advantageous in that we need not locate the "easiest" mapping of the function, only a mapping that is easier than those explored previously.

The criteria for adjusting the size of the hypercube imposed on the solution space is a function of the current population diversity and the previous interim solution. No complex mechanisms are required for these adjustment functions, although additional parameters are used. The use of shorter strings to perform search in a reduced hypercube allows the employment of smaller populations. This decreases the execution time for GENITOR [12].

A. Advantages of Relocating the Subpartition Sample

The reduced hypercube induced by delta coding produces a more focused search. It also introduces the possibility of excluding the global optimum from the reduced search space. However, parts of the solution space that are excluded from the reduced search space in one delta iteration can be reintroduced later in the search. Each new interim solution repositions the hypercube being searched, allowing the algorithm to effectively "crawl" along a rough fitness surface in search of better solutions. Figure 3 shows a 1-dimensional conceptual view of this behavior when minimizing a function. The sampling window has been reduced to some lower bound to avoid confusion between the movement of the sampling window and its expansion/contraction. The sampling window converges on new solutions, allowing relocation of the hypercube to search new portions of the problem space.

The parameters in a delta iteration are decoded as delta values ($\pm\delta$) and added to the interim solution. The resultant parameter values must be contained within the original search space. Therefore, a new hypercube imposed around an interim solution close to boundaries in the solution space will wrap around on the opposite edges. This is also illustrated in Figure 3.

An additional advantage of delta coding is the ability to add bits back into the delta parameter representations. If the parameter reduction mechanism has reduced the sampling window such that the search cannot converge on a new solution, the algorithm will not be able to escape that subpartition of hyperspace. This effectively stalls the search. Adding bits back into the delta parameter representations enlarges the sampling window. This allows the algorithm to recover from aggressively focusing the search in smaller subpartitions of hyperspace (Figure 4).

IV. Algorithm Descriptions

The mutation driven stochastic hill-climbing algorithm used in these comparisons randomly generates an initial binary string. Then, search is performed by applying mutation to that string. The changes produced were kept whenever the resulting string was better than or equal to the string before mutation. This process is repeated until the optimal solution is found or until some maximum number of trials is executed.

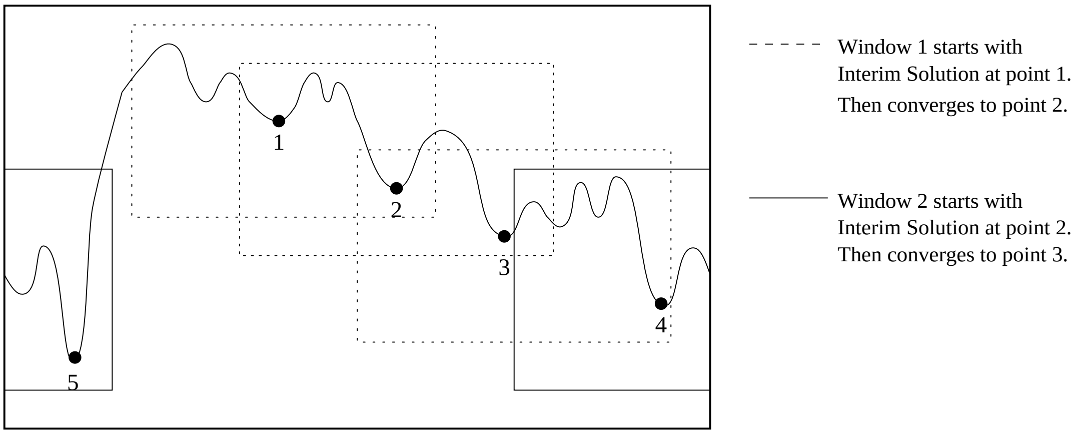

Figure 3: Crawling along a Rough Fitness Surface. Assuming the search has converged to the interim solution at point 1, the subpartition of hyperspace defined by window 1 is searched. This allows the search to converge to point 2. The subpartition of hyperspace surrounding point 2 is then searched, converging on point 3. The search eventually converges on point 5.

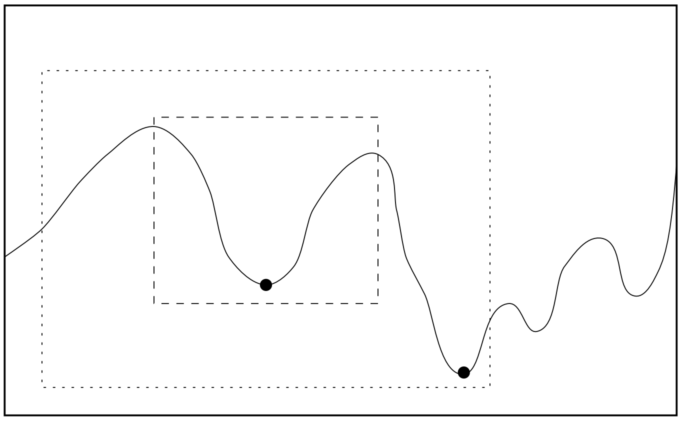

Figure 4: Expanding the Delta Coding Search Window. Expansion is performed after converging to the same interim solution (i.e. 1 & 2) on consecutive iterations using a reduced search space string representation.

The CHC Adaptive Search Algorithm [4], is a generational genetic search model. However, this algorithm differs from the standard genetic algorithm in several ways and seems to be designed such that all of its subcomponents are necessary to achieve the best performance. First of all, the CHC algorithm employs *heterogeneous recombination* as a method of "incest prevention" [4]. The operator designed for this task performs uniform crossover over half of the differing bits (HUX). HUX is a very disruptive crossover operator designed to sample new points in the search space by ensuring that parents paired for recombination have at least some appreciable differences. If the number of differing bits exceeds some threshold, uniform crossover is applied to half of those positions, where the positions are chosen randomly. Additionally, the CHC algorithm does not bias the selection of strings for reproduction in favor of those strings with a higher fitness. Instead the algorithm employs a cross-generational selection/competition mechanism, producing offspring by randomly and uniformly choosing strings for recombination from the parent population. The best N strings, from the parent and offspring populations are selected to form the population for the new generation, where N is the population size. No mutation is applied during the recombination phase of the algorithm.

Like delta coding, CHC uses a form of restart to infuse new diversity into the population with which to continue genetic search. CHC's restart mechanism is known as *cataclysmic mutation*. When no offspring can be inserted into the population of a succeeding generation, the best individual in the population is used as a template to reinitialize the population. The new population includes one copy of the template string. The remainder of the strings in the new population are formed by repeatedly mutating some percentage of bits in the template string and copying the result into the new population. This creates a population that is biased toward a good solution but with new diversity to sustain further search. While this mechanism preserves the progress made so far and allows continued search it does nothing to change the mapping of the strings with respect to one another in Hamming space.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Percent Solved	100%	100%	100%	30%	100%	100%	100%	3%
Average Trials	507	19591	767	58067	481	104844	115267	227701
Maximum Trials				100000				500000
Average Best				-2.038				0.099
Mutation Rate	0.04	0.15	0.04	0.04	0.07	0.02	0.05	0.02

Table 1: Mutation Driven Stochastic Hill-Climbing Algorithm Performance Using Gray Coding.

Algorithm	Problem	F1	F2	F3	F4	F5	F6	F7	F8
	Percent Solved	20%	23%	100%	20%	40%	00%	40%	00%
Stochastic Hill Climber	Average Trials	384	27672	544	62284	884		346667	
	Maximum Trials	200000	100000		100000	50000	500000	500000	500000
	Average Best	0.0001	0.0002		-1.893	1.16	6.71	-4170.72	0.182
	Mutation Rate	0.05	0.09	0.04	0.03	0.08	0.03	0.05	0.03
	Percent Solved	100%	100%	100%	100%	100%	00%	100%	23%
CHC	Average Trials	56892	37737	687	14715	4443		15230	345242
	Maximum Trials						500000		500000
	Average Best						0.18		0.038
	Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
	Population Size	15	25	15	25	50	800	100	200
δ Coding	Average Trials	674	2365	521	4984	1204	250005	22852	48806
	Linear Bias Selection	2.00	2.00	2.00	2.00	2.00	1.90	1.90	2.00
	Mutation Rate	0.04	0.02	0.05	0.04	0.06	0.01	0.02	0.005

Table 2: Performance of the Algorithms Without Gray Coding.

V. Algorithm Performance Comparisons

The test functions for the comparisons presented here included the DeJong test suite (F1 - F5) [3, 5] and the Rastrigin (F6), Schwefel (F7) and Griewank (F8) functions [10]. Previous genetic algorithm performance comparisons using these functions report the use of Gray coding to enhance performance [4, 11]. Therefore, we tested the ability of the stochastic hill-climbing algorithm to locate the optimal solution on these functions using Gray coding. A wide range of mutation rates were tested, with the best results being reported in Table 1.

In Table 1, *percent solved* refers to the percentage of those runs that found the optimal solution.[2] *Average trials* indicates the number of recombinations necessary to locate the optimal solution averaged over those runs that were successful. The *average best* and *maximum trials* values are recorded only when the optimal solution is not found 100% of the time. The average best value is the average fitness of the best individual in the population for all 30 runs after the maximum number of trials have been executed. The *mutation rate* is the probability that a bit is complemented.

Table 1 shows that when Gray coding is employed, the stochastic hill-climbing algorithm consistently finds the optimal solution for 6 of the 8 test functions. Additionally, the stochastic hill-climber is able to solve many of these functions faster than many genetic algorithms, as reported in other studies [4, 11]. However, many of these test functions were designed specifically to be difficult for hill-climbers. This suggests that Gray coding may be capable of masking some of the attributes of these functions that make them difficult.

While Gray coding eliminates Hamming cliffs in the search space it also alters other attributes of a function that are relevant to search in Hamming space. Gray coding has been shown to change the number of local optima in the search space as well as the relative size of the attraction basins [9]. Gray coding also produces a more dramatic transformation with respect to the genetic algorithm: the subsets of strings found in the hyperplane partitions represented by schema also change.

Assuming that Gray coding does indeed compromise this set of test functions by altering those attributes that were incorporated to make them difficult, then it should not be used for genetic algorithm comparative studies. Any mapping which allows the function to be solved by simple stochastic hill-climbing compromises comparative studies, since those algorithms that exploit hill-climbing have an advantage. We would also argue that genetic search is most needed on problems where hill-climbing algorithms fail. Thus, comparisons between genetic algorithms should utilize more difficult test functions. Therefore, the performances of the mutation driven stochastic hill-climbing algorithm and the CHC and delta coding genetic algorithms were compared on the same test functions without using Gray coding (Table 2).

For the comparisons in Table 2 the cataclysmic mutation rate (35%), population size (50) and HUX crossover threshold parameters for the CHC algorithm were set as

[2] Several methods have been reported for determining the optimality for F4. Here, we have used the evaluation value of -2.5 to determine when optimality has been reached.

suggested by Eshelman [4]. The delta coding algorithm used two point reduced surrogate crossover [2].

When Gray coding is not applied to the functions in the test set, the stochastic hill-climber can solve only one of the test functions consistently (F3). Even the CHC genetic algorithm cannot consistently solve the F6 and F8 test functions when Gray coding is not used. Thus, when a binary encoding is used, these problems are much more difficult to solve.

VI. Conclusions

We have shown here that a mutation driven stochastic hill-climber performs better than genetic algorithms on several of these test functions when Gray coding is used. However, simple stochastic hill-climbers do not always perform well on real-world problems and genetic algorithms are most needed when simpler methods fail. Therefore, we would argue that a good test suite should be resistant to simple stochastic hill-climbing algorithms.

Additionally, these results lead to the conclusion that the suite of test problems used here, although currently serving as the genetic algorithm community benchmark, need to be reviewed. Many of these functions were never intended to be lasting genetic algorithm benchmarks [3] and there is a real need for a new and more difficult test suite for genetic algorithm performance comparisons.

We also question the use of Gray coding in comparative work when using test functions that were designed to be difficult in either numeric or standard binary space. Gray coding dramatically alters key features of the function, possibly transforming the function into an easier function mapping where traditional methods are able to perform better than genetic algorithms. However, when the functions are not Gray coded the genetic algorithms tested here perform much better than the mutation driven stochastic hill-climbing algorithm. While the stochastic hill-climber is able to consistently solve only a single function, the CHC algorithm is able to find the optimal solution for 6 of the 8 test functions and delta coding consistently solves all of the test functions. This may also imply that the role of mutation in the genetic algorithm is changed when Gray coding is used on these test functions. Instead of simply providing a method for diversity management, mutation may play the role of a hill-climber and actually aid in solving the function.

Finally, these results represent an initial performance comparison of delta coding on standard benchmark functions (A more complete comparison is given in [8]). Delta coding performs better than the other algorithms compared here in all but a single case when Gray coding is not applied to the test functions. This indicates that delta coding is able to adaptively locate a mapping of the search space that is "easier" for the genetic algorithm to solve.

Acknowledgements

Our thanks to Raja Das, John Dzubera and Scott Gordon for making suggestions along the way and reading earlier drafts of this work.

References

[1] James Baker. Adaptive Search Methods for Genetic Algorithms. In John Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 101–111, 1985.

[2] Lashon Booker. Improving Search in Genetic Algorithms. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 5, pages 61–73. Morgan Kaufmann, 1987.

[3] Ken DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Department of Computer and Communication Sciences Ann Arbor, Michigan, 1975.

[4] Larry Eshelman. The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[5] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[6] David Goldberg. Sizing Populations for Serial and Parallel Genetic Algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79. Morgan Kauffman, 1989.

[7] Keith Mathias and L. Darrell Whitley. Remapping Hyperspace During Genetic Search: Canonical Delta Folding. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms - 2*, pages 167–186. Morgan Kaufmann, 1993.

[8] Keith E. Mathias and L. Darrell Whitley. The Delta Coding Algorithm: Comparisons to CHC, SGA, GENITOR and Hill Climbing. Submitted for publication.

[9] Keith E. Mathias and L. Darrell Whitley. Transforming the Search Space with Gray Coding. Submitted for publication.

[10] H. Muhlenbein, M. Schomisch, and J. Born. The Parallel Genetic Algorithm as Function Optimizer. In L. Booker and R. Belew, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 271–278. Morgan Kauffman, 1991.

[11] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kauffman, 1989.

[12] Timothy Starkweather, L. Darrell Whitley, and Keith Mathias. Optimization Using Distributed Genetic Algorithms. In H.P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature*, pages 176–185. Springer/Verlag, 1990.

[13] L. Darrell Whitley. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kauffman, 1989.

[14] L. Darrell Whitley, Keith Mathias, and Patrick Fitzhorn. Delta Coding: An Iterative Strategy for Genetic Algorithms. In L. Booker and R. Belew, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84. Morgan Kauffman, 1991.