Delta Coding: An Iterative Search Strategy for Genetic Algorithms

D. Whitley and K. Mathias
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523

P. Fitzhorn
Mechanical Engineering Dept.
Colorado State University
Fort Collins, CO 80523

Abstract

A new search strategy for genetic algorithms is introduced which allows iterative searches with complete reinitialization of the population preserving the progress already made toward solving an optimization task. *Delta coding* is a simple search strategy based on the idea that the encoding used by a genetic algorithm can express a distance away from some previous partial solution. Delta values are added to a partial solution before evaluating the fitness; the delta encoding forms a new hypercube of equal or smaller size that is constructed around the most recent partial solution. Results are presented on two optimization problems involving geometric transformations; solving these problems with precision is difficult for conventional genetic algorithms as well as traditional mathematical optimization techniques. Tests using single population and distributed genetic algorithms are compared to *delta coding*. *Delta coding* is shown to produce more precise solutions while reducing the amount of work necessary to reach the solution.

1 INTRODUCTION

Genetic algorithms have been applied to many types of optimization problems. However, the limited success experienced on some problems, especially those requiring precise solutions, has driven researchers to look for new and more powerful strategies to augment the performance of genetic algorithms. *Delta coding* is introduced and shown to be capable of producing improved performance by yielding more precise solutions with less work. *Delta coding* treats population strings not as parameters, but as *delta values* ($\pm\delta$) which are added to a partial solution before being evaluated. The first run of a genetic algorithm (GA) using delta encoding is like any other run of the genetic algorithm. Subsequent runs are made by using the best solution obtained on the most recent run as a partial solution; the genetic algorithm is restarted with the substring coding for each parameter representing a distance or delta value ($\pm\delta$) away from the value of the corresponding parameter in the most recent partial solution forming a new hypercube around the previous solution. The "delta hypercube" can also be reduced or enlarged each time the genetic algorithm is restarted.

Delta coding effectively avoids what is usually a difficult trade off between achieving fast search and sustaining diversity (and thereby avoiding premature convergence). Diversity is a problem because the longer a genetic algorithm runs and the closer it gets to a solution, the less search power it has. *Delta coding* avoids the need to sustain diversity, since diversity is reintroduced by generating an entirely new and random population; previous work is preserved because the hypercube corresponding to the new encoding is defined with respect to the previous partial solution. By using smaller populations, *delta coding* reduces the amount of work it takes to reach a solution while at the same time improving the precision of the solution. Furthermore, if the precision of the resulting solution is more critical than search time, search can continue indefinitely without worrying about premature convergence, while normal genetic algorithms must be completely restarted when convergence occurs without reaching an adequate solution.

Delta coding preserves hyperplane sampling (and the theoretical foundations of genetic algorithms) since each individual iteration is a single run of a genetic algorithm; only the mapping strategy has changed. *Delta coding* makes mutation unnecessary. Furthermore, the "delta hypercubes" defined with respect to different partial solutions are never the same. Thus, Hamming cliffs are moved to different pairs of strings in the search space each time the hypercube is redefined. This has the potential to eliminate gray coding and may also be helpful in solving deceptive problems.

Delta coding is tested on two geometric transformation problems. The performance of GENITOR, GENITOR II, and *delta coding* are examined. While we use *delta* *coding* in conjunction with GENITOR, *delta coding* is an encoding strategy that can be used with any genetic algorithm. The two transformation problems involve a search for the set of optimal geometric transformation parameters (i.e. rotation, scale and/or translation) that will superimpose a set of test points in one geometric domain onto a given set of fixed points in another. The first problem can be stated as follows: Given two independent two dimensional representations of a single object, find a transformation which maps one representation (the primary image) onto the other (target image). The transformation is achieved by reducing the sum of the squared errors on two sets of n landmark points, one set of n from the primary image, and a similar set of n from the target, which must match when the two images are properly superimposed. In the second problem, a three dimensional primary representation must be mapped onto a target two dimensional image. As in the first problem, the error is minimized over a set of landmark points from the fixed two dimensional image.

2 BACKGROUND

Genetic algorithms work with a (typically binary) string encoding of a problem's parameters. It also employs an evaluation function to determine the relative performance of any particular set of parameters. The selection of strings for replication and reproduction is based on the performance measure. A string of parameters must be encoded in such a way that they can be broken apart and combined with other strings to form new parameter sets, which are referred to as offspring. The fitness (i.e. performance) of strings is used to allocate reproductive trials, such that a higher fitness value translates into an increased probability that a particular parent will be chosen to recombine with another parent string during reproduction. Recombination takes place between two parent strings by identifying two points randomly on the strings and choosing the information between the points from one parent to pass on to the offspring and the information outside of the points from the other parent to pass on to the offspring. The reduced surrogate crossover operator [2] used in these experiments, can ensure that offspring are not duplicates of the parents.

The goal of genetic algorithms is to exploit hyperplane information feedback by evaluating the relative fitness of strings, and then use this information to allocate reproductive opportunities so as to direct the search toward particular regions or partitions in hyperspace that contain above average competitive solutions. This method of search has been shown to be an effective optimization technique; genetic algorithms may be superior to other mathematical methods of search if the problem is complex with multiple local minima or if gradient information is difficult to obtain.

Sustaining diversity is a pervasive problem with genetic algorithms. Several strategies have been developed to support diversity maintenance such as mutation, adaptive mutation and distributed genetic algorithms[12] as well as more divergent variations on genetic search such as Ackley's connectionist genetic hill-climber [1] and Shaefer's ARGOT Strategy [10]. Diversity maintenance is critical, since the search reaches a plateau and stops if the entire population has converged to a single solution (or is populated with strings whose potential offspring are already members of the population). A typical search with a genetic algorithm shows significant progress toward better solutions when the population exhibits a large degree of diversity, but as search continues, progress becomes asymptotic to a limit imposed by the lack of diversity in the remaining population.

When first developing the *delta coding* idea, we did not relate it to any existing method. The motivation was based on the fact that solutions found by genetic algorithms were sometimes close to a precise solution such that we could complement a small number of bits and obtain a globally optimal solution. Also, recent theoretical work [3] shows that selective pressure on hyperplanes is not uniform during genetic search; bits that are selected against early on may actually be preferred as the search narrows to concentrate on specialized regions in hyperspace. But selective pressure cannot work during the later stages of search if there is not enough diversity in the population to allow alternate bit combinations to be tested.

After developing *delta coding*, we realized that it shared similarities with other strategies, especially ARGOT and Dynamic Parameter Encoding (DPE). Other strategies that have similarities that are not as obvious at first glance include micro-GAs. The next section briefly reviews GENITOR and the distributed GENITOR II, since these are compared to *delta coding*. ARGOT, DPE and micro-GAs are also discussed. Future work should include comparisons with these algorithms and should examine the merger of subsets of these strategies with *delta coding*.

2.1 THE GENITOR ALGORITHM

GENITOR is a genetic algorithm that uses "one at a time" recombination between two parents to form an offspring; each offspring is immediately evaluated and placed in a ranked population, displacing the string in the population with the worst fitness. This is different from the mass reproduction found in standard genetic algorithms where offsprings displace parents regardless of the fitness values of either string. One at a time reproduction guarantees that the best solutions found so far will be held undisturbed in the population until a better solution is located. GENITOR has displayed superior performance in many problem domains [15]. The perpetuation of superior strings guides the search by providing more bias or probability of commitment toward better solutions thereby introducing a kind of hill-climbing to genetic search, with only improved points in the search space being permitted to become part of the population. This can potentially be detrimental to the search in some instances since sampling biases can result from early commitment to strings that may not be representative of the general distribution of strings in hyperspace. Still, no one string dominates search and it has been shown that GENITOR does allocate increasing reproductive trials to hyperplanes in the population that contain above average strings [12].

2.2 GENITOR II: A DISTRIBUTED GA

GENITOR II is a version of GENITOR that attempts to improve search and maintain diversity by using distributed populations. Small populations, termed subpopulations, represent an independent search except that the subpopulations rendezvous at fixed intervals and exchange information by swapping copies of their "best" strings. The exchange interval, subpopulation size, number of subpopulations, and number of strings exchanged are user defined [12].

The algorithm used to exchange strings between subpopulations is a directional adjacent ring neighborhood distribution. This algorithm allows the subpopulation at position 1 to distribute copies of it's best n strings to the neighbor at position 2 at the first rendezvous. The new strings in the subpopulation at position 2 displace the n worst strings. This distribution occurs at all adjacent subpopulations in the ring at the same synchronized interval. During the next distribution the subpopulation at position 1 distributes copies of it's n best strings to the subpopulation at position 3, with corresponding distribution occurring throughout the ring. This method of distribution allows the best string from the subpopulation at position 1 to reach all positions in $log(X) + 1$ distributions, (where X is the number of subpopulations in the ring), assuming the string remains the best for the number of trials needed to perform $log(X) + 1$ swaps.

The GENITOR II algorithm has been shown to yield better results than GENITOR and standard GAs on many classes of problems. The distribution of the n best strings from a subpopulation provides a method to sustain diversity, share information and correct for sampling biases that may occur in individual populations. The GENITOR II algorithm not only shows improved performance in terms of more accurate solutions, it also requires less overall work [11].

2.3 THE ARGOT STRATEGY AND DYNAMIC PARAMETER ENCODING

The ARGOT strategy (Adaptive Representation Genetic Optimizer Technique) combines many ideas to improve optimization; the results appear to be successful, but the strategy is complex. The ARGOT strategy is designed to solve problems using a genetic algorithm, but several environmentally triggered operators are added to alter intermediate mappings that drive the search. Binary strings are used to represent search parameters. However, an intermediate mapping is performed between the bit strings and the search space. These intermediate mappings are based on internal measurements such as parameter convergence, parameter variance and parameter "positioning" within a possible range of parameter values. These measurements are used to drive strategies within ARGOT such as adjusting parameter resolution and roving parameter boundary locations [10].

Dynamic parameter encoding (DPE) is a strategy explored by Schraudoph and Belew [9] which has many similarities to ARGOT; the accuracy of the encoded parameters are dynamically adjusted to increase the resolution of the answer and to zoom in on the most promising area of the search space. DPE is designed to reduce the affects of the arbitrary experimenter assigned resolution inherent in solutions involving genetic optimization. At first the string encodes only the most significant bits of each parameter (e.g., 4 bits). This represents a coarse grain partitioning of the function space. As the genetic algorithm begins to converge, the most significant bit is recorded, dropped from the encoding, and a new bit is introduced which adds additional precision and creates a finer grain partitioning to the search space. Thus, while the number of bits used by the genetic encoding remains constant, the function is searched using an increasing level of detail.

Both of these approaches clearly have similarities to *delta coding* because bits are added or deleted (or remapped) as the range and location of parameter boundaries change. *Delta coding* does not allow each parameter to change independently (perhaps this could be done) while on the other hand ARGOT and DPE do not have discrete stages of search as does *delta coding.* The discrete stages of search may in fact be advantageous; the other strategy we have looked at that uses discrete stages of search is micro-GAs.

2.4 MICRO-GAs AND OTHER REINITIALIZATION METHODS

The micro-GA [7] uses a small population (e.g. five strings). The population is measured for convergence either by genotype convergence or phenotype convergence. If the population has converged the best string is kept and the rest of the population is randomly regenerated and the search is repeated. No mutation is used. This strategy has been shown to give good performance on certain test problems. Goldberg initially described this method of implementation for serial GAs [5]. The motivation is to maximize the number of schemata sampled during genetic search. Cataclysmic mutation, as described by Eshelman[4] also provides a method of restarting the GA search by reinitializing the population. During reinitialization, the best string in the population at the time of convergence is used as a template by randomly complementing some proportion of the bits; the resulting strings are used to reinitialize the population.

Micro-GAs and these other strategies are somewhat like *delta coding* in that the population is reinitialized and prior results are included in the population by saving the best string. One advantage of delta coding is that a completely new hypercube is defined, and the size and location of the new hypercube can be varied. One disadvantage of micro-GAs, for example, is that the small population prevents adequate hyperplane sampling and simple hill-climbing may dominate the search.

3 DELTA CODING

We now look at *delta coding* in more detail. Strategies such as micro-GAs are similar to *delta coding* in that diversity is *regenerated* in order to sustain search. Other strategies like ARGOT and DPE are similar in that they focus on promising areas of search. One of the advantages of *delta coding* is its simplicity: it is easy to understand, easy to implement and easy to analyze given the existing theory of genetic algorithms. Like ARGOT and DPE, *delta coding*, remaps hyperspace, can decrease the size of the search space by restricting the range of the *delta values* ($\pm\delta$) at each iteration and additionally eliminates diversity concerns.

Our *delta coding* implementation uses GENITOR as its basis. On the initial run, GENITOR evaluates the problem in a normal fashion except that the diversity of the population is checked on each trial. The population diversity is monitored by measuring the Hamming distance between each of the parameters in the best and worst strings in the population. If the sum of the parameter Hamming distances is more than one, the search continues. If the distance is less than or equal to one, the diversity is defined to be *sufficiently exploited* and the iteration is stopped. The best solution is saved as a starting point for the next *delta iteration.* The new values encoded on the string are recoded with a sign bit, thus defining a positive or negative delta value. Subsequent delta runs will often explore a smaller solution space. The number of bits used to encoded each delta value is reduced by a user determined number of bits. The population is reinitialized and the GENITOR algorithm is allowed to resume evaluation. Each delta parameter, when decoded for fitness evaluation, is applied as a delta value ($\pm\delta$) to the parameters saved from the previous iteration. This produces a refined search in a reduced hypercube space centered around the best initial solution.

```
PHASE 1:  /* Apply GENITOR in Normal Mode */
    Apply recombination/Evaluate/Insert Offspring
    Check Hamming Distance Between Best and
       Worst Members in Population
    IF (Hamming Distance > 1) Continue PHASE 1

TRANSITION PHASE:
    Save Best Solution of Population as PARTIAL
    Re-initialize Population
    Encode Parameters Using (X - 1) Bits
              (Plus Extra Bit Used as Sign)

ITERATION PHASE:
   DELTA:  /* Apply GENITOR in Delta Mode */
       Apply recombination
       Add Decoded String Value to PARTIAL
       Evaluate/Insert Offspring
       Check Hamming Distance Between Best and
            Worst Members in Population
       IF (Hamming Distance > 1) Continue DELTA

   IF (Maximum Trials > Allocated Trials) OR
     (PARTIAL < Threshhold) HALT
   ELSE
     Save New Best Solution as PARTIAL
     Re-initialize Population
     IF (Delta Values = 0) THEN
          Encode Parameters with 1 Extra Bit
     ELSE
          Encode Parameters with 1 Less Bit

   Continue ITERATION PHASE
```

Table 1: *Delta Coding Algorithm*

Each *delta iteration* is allowed to proceed until diversity has been sufficiently exploited. The goal is to achieve solution improvement while the population is diverse. Typically, as a genetic algorithm begins to converge, more and more effort is expended to obtain smaller and smaller reductions in error. When using delta coding we want to stop search when the probable payoff for continued search begins to decline. In actual practice, this is not easy to determine, but the exact stopping criteria need not be too precise as long as the search is not stopped too quickly.

Once an iteration is terminated, the best result from that iteration is saved for the next *delta iteration.* Typically, at each iteration the *delta coding* is reduced by a single bit for each parameter; reductions are limited to some minimal number of bits for the *delta values* (e.g. 4 to 6 bits). If a delta run converges to exactly the same solution (i.e., all zeros), the number of bits used to encode the *delta values* ($\pm\delta$) is increased, thus expanding the hypercube. This search process continues until a solution meeting the user defined criteria is found or until the maximum number of trials (defined by the user) for the entire collection of iterations is exhausted. The algorithm is shown in Table 1.

One critical finding (that is somewhat counter-intuitive) is that *delta coding* works best when defined with respect to the immediate previous solution, *not the best solution found so far*. The reasons for this are not completely clear; however, a different (inferior) solution indicates something about the distribution of strings in hyperspace and provides a new direction in which to search. It is also consistent with the notion that search should not be exclusively driven by the single best string found so far. Furthermore, this strategy could help the search to escape local minima without having to expand the delta encoding.

4 PROBLEM DESCRIPTIONS

We attempted, unsuccessfully, to find techniques in traditional mathematical optimization methods that could be used to find optimal rigid body transformations for superimposing a set of test points onto a set of fixed target points. The only methods which appeared to be applicable for our 3-D problem were brute force methods that involved a great deal of ad hoc search. While we have not looked at methods such as simulated annealing, so far the genetic algorithm is the only approach that has produced adequate results using modest computational resources.

In the 2-D problem, four points are defined in two dimensional space and used as target points. Five transformation parameters are then applied to another set of four corresponding points in two dimensional space (the primary image) with the goal of overlaying them on the target points. The five transformation parameters used were translation along the X and Y axis, scale in the X and Y directions, and rotation around the Z axis (restricting the points to the X-Y plane during rotation). A further constraint was imposed on the transformation in that each point was identified with a label such that *p1* of the primary image must overlay *p1'* of the target image for the transformation to be correct. The parameter set was encoded in binary using eight bits for each parameter, for a total of 40 bits. The fitness function was designed to measure the Euclidean distance between the corresponding target and transformed points, summing the squared distances for all point pairs. The goal was to minimize the error. (For implementation details see [8]).

The 3-D to 2-D transformational problem was defined by a specific application and motivated the 2-D test problems [6]. The goal in the 3-D transformation problem was to provide a set of transformation parameters that would map a set of 3-D points onto a target set of points in a 2-D domain. As in the 2-D transformation problem, the points were tagged so that the points undergoing transformation corresponded to specific 2-D target points. The evaluation function was used to measure Euclidean distances and report the sum of the squared distances as the error.

For the 3-D problem there were only three points on the object representation available to provide error information and eight transformation parameters were required [6]. The eight parameters were translation along the X, Y, Z axis, rotation around the X, Y, Z axis, a single uniform scale, and perspective. Parameters were encoded using 12 to 14 bits each, for a total of 104 bits. (For implementation details see [8]).

5 EXPERIMENT DESCRIPTIONS

The 2-D transformation problems were developed as a smaller test environment to see how genetic algorithms would perform on this class of problems. Several variants of the 2-D problem were developed and set up to run under GENITOR; each variant was known to have a solution of zero error. Several attempts at tuning parameters for optimal performance were performed. The problems were also run in the GENITOR II environment. The tunable parameters were kept the same in the GENITOR II runs as in the GENITOR runs except the population size and number of trials allocated to each experiment were altered to adjust for the use of multiple subpopulations. There were ten subpopulations used in the 2-D experiments. The swap interval was set to 5,000 and the number of strings distributed at each interval was five. With less than optimal results using GENITOR II the problem was encoded using the *delta coding* strategy. An effort was made to tune the population size and maximum overall trials to the lowest values possible while still achieving superior results. The bias value was kept at 2.0 in an effort to maintain an agressive level of search. Each experiment was repeated 30 times to obtain an adequate comparative sample.

5.1 REDUCTION AND EXPANSION STRATEGIES

Several types of reduction schemes were tested for use with *delta coding*. These reduction strategies involve user tunable values including (but not limited to) the smallest number of bits for a parameter during a run and the number of bits by which to reduce or expand each parameter during each new iteration. We tested numerous reduction and expansion strategies; only key examples used on the 2-D problem are reported here [8]. First, we reduced each parameter by 1 bit at each iteration, until the *delta coding* used only 2 bits. We found however, that we obtained better results if the *delta coding* was not allowed to use less that 4 bits; in other words, after reducing the *delta coding* to 4 bits per parameter, it stays at four bits (unless we expand the encoding). We thought that it was perhaps the 4 bit limit rather than the 1 bit reduction that was important, but this proved not to be the case in our test problems. We tried reducing by 2 bits with a 4 bit limit to the minimal encoding; this did not perform as well. Reducing by 4 bits to more quickly reach the 4 bit limit further degraded performance.

Simple 1 bit reduction per parameter proved to be the best strategy on both the 2-D and 3-D problems; however, on the 3-D problem a 6 bit limit produced the best results, while on the 2-D problems the best results were obtained with a 4 bit limit. Solution curves for a few of the *delta coding* reduction strategies are presented in Figure 2. These results suggest that a conservative strategy which does not commit to a solution too quickly has an advantage over other strategies.

The other side of this issue is *delta coding* expansion. In our implementation of *delta coding* the criteria used to determine when the hypercube should be expanded was as follows: if a run converged to zero such that the solution was the same as that obtained on the previous iteration, then the hyperplane is enlarged by increasing the number of bits used in the *delta coding*. This "opens up" the search space when 1) further improvements seem unlikely or 2) the space defined by the current hypercube somehow favors the previous solution. The encoding is not expanded if the search converges to an inferior, but different solution. If a different solution is found, inferior or not, the encoding is reduced using the new inferior solution as the partial solution for delta encoding. This decision is based on empirical results and the intuition that a different but inferior solution shifts the direction of search.

6 RESULTS

Comparison of different approaches to the 2-D mapping problem are provided in Table 2 with performance curves in Figure 1. Several versions of the 2-D to 2-D problem were examined; on most of these "test cases" *delta coding* found zero error 100% of the time. Table 2 reports the worst test case results for *delta coding*, while it reports best test case results for GENITOR and GENITOR II. *Delta coding* out performs all other implementations compared in this context and provides a reliable mechanism to find a globally optimal set of transformations with zero error.

A summary of comparison experiments for the 3-D to 2-D transformation problem is provided in Table 3. The corresponding solution curves are presented in Figure 3. The swap interval for the GENITOR II experiments was set to 2500 and the number of strings distributed was set to five. The total number of trials allocated for each of the 3-D experiments was 500,000; for example, with 10 subpopulations the number of trials is 50,000 for each of the individual subpopulations in the GENITOR II environment. The population size for the 3-D problem

Genetic Method	Pop. Size	Sub-Pop.	Bias Value	Max. Trials	Avg. Error	Percent Solved
GENITOR	3,700	1	1.8	100,000	0.083	10%
GENITOR II	300	10	1.8	20,000	0.045	10%
GENITOR II	1,000	10	1.8	100,000	0.003	80%
Delta Coding	200	1	2.0	20,000	0.001	93%

Table 2: *Comparison of GENITOR and GENITOR II to* Delta Coding *for the 2-D Problem. Worst case results are reported for* delta coding*, while best case results are given for GENITOR and GENITOR II.*

Genetic Method	Pop. Size	Sub-Pop.	Bias Value	Max. Trials	Avg. Error	Best Found
GENITOR	5,000	1	1.75	500,000	1.45	0.4900
GENITOR II	500	10	1.75	500,000	1.22	0.3270
GENITOR II	50	100	1.75	500,000	0.88	0.1950
Delta Coding	500	1	2.00	500,000	0.10	0.0037

Table 3: *Comparison of GENITOR and GENITOR II to Delta Coding for the 3-D to 2-D Problem. For GENITOR II population size refers to subpopulation sizes.*

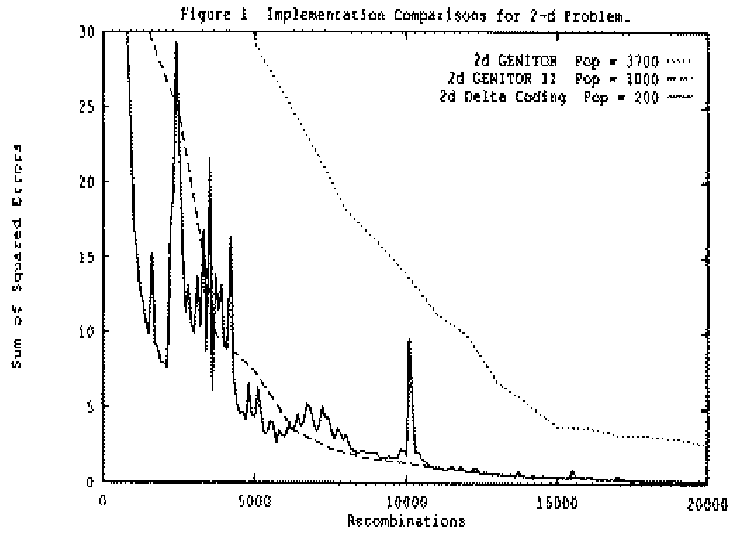
was set to 500.

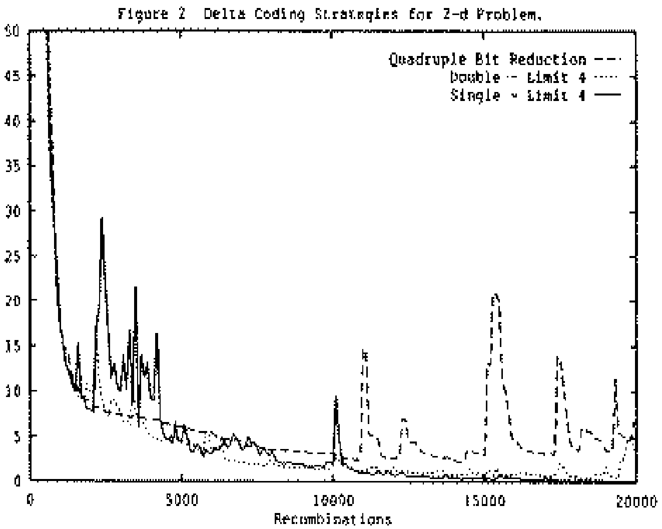
The delta strategy appears to out perform the other implementations on the 3-D mapping problem; it converged faster using an overall population size that is 10% the size of the others.

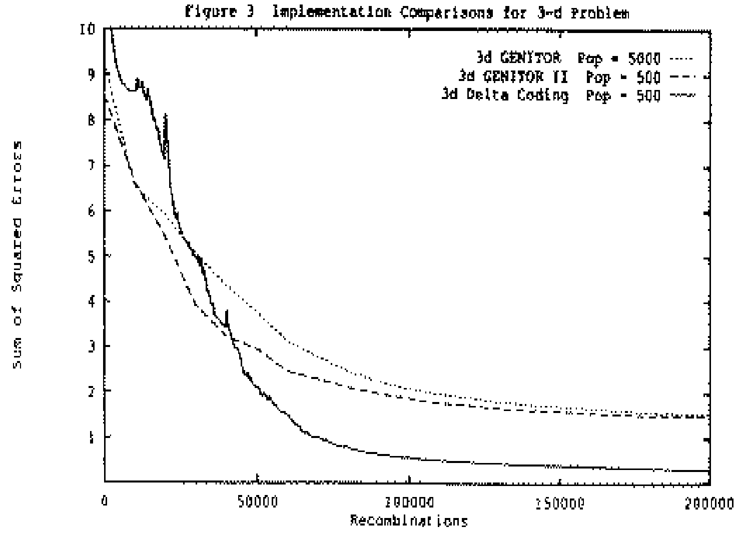
Figure 3 also shows that *delta coding* produces these more precise results using fewer trials; the *delta coding* experiments were largely converged by 100,000 trials. Furthermore the use of smaller population sizes in conjunction with *delta coding* in the GENITOR environment results in significant time savings [11]. The optimal solution was known to be an error of zero in all test cases and yet none of the experiments yielded a single run with an error of exactly zero. *Delta coding* made little progress after the initial 100,000 recombinations; ideally delta coding should have allowed the algorithm to continue to make progress, but in this case it did not. Nevertheless, the results are encouraging.

7 CONCLUSIONS

Delta coding has several attractive properties. It is conceptually simple to understand and simple to implement. It does not rely on forced mechanisms such as mutation to sustain diversity; it simply continues to search until the diversity in the current iteration is exhausted or reduced. It then saves the best solution parameters and reinitializes the population. This provides a new population with which to search again. This resurgence of diversity for each *delta iteration* functions to maintain a vigorous search at all times.







Figures 1-3. Figures 1 and 3 compare performance for GENITOR, GENITOR II, and GENITOR with delta coding on the 2-D to 2-D and the 3-D to 2-D mapping problems. Figure 2 compares performance for GENITOR with delta coding on the 2-D to 2-D problem using different reduction strategies. The "spiking" behavior in all three figures represents restarts of the algorithm using delta coding which are accompanied by changes in the encoding. The graphs do not represent the entire search; scales where choosen to highlight the "spiking" behavior and the early convergence of GENITOR using delta coding. Early "spiking" from vigorous restarts (such as that characterizing the single bit reduction in Figure 2) typically produces better optimization. The "spiking" behavior in Figure 3 is somewhat suppressed by the scale of the figure.

With each *delta iteration* the number of bits used to encode the parameters are typically reduced and thus, the solution space is made smaller. Since the parameters are *delta values* ($\pm\delta$) the new hypercube remaps the space, wrapping around on the opposite extreme of the parameter ranges and effectively remaps Hamming cliffs. The criteria for determining when to collapse the hypercube imposed on the solution space is a function of current population convergence.

Another advantage of *delta coding* is that smaller population sizes than those used in other implementations can be used; this can translate into significantly less work [11]. The reduction of the search space also potentially helps to reduce search time.

The failure to locate the optimal solution in the 3-D mapping problem indicates that there is more work to be done. Although these results are encouraging there are numerous ways in which *delta coding* might be enhanced. Some potential difficulties with *delta coding* are 1) additional user tunable parameters, 2) the definition of hyperplane expansion criteria, and 3) developing efficient but effective methods for determining when the population diversity has been *sufficiently exploited.*

While we have not yet carried out experiments to test these ideas, we also believe that *delta coding* may help in solving two types of problems that are known to cause difficulties for genetic algorithms. First, *delta coding* may help to solve deceptive problems. Deceptive problems exist because the relationship between specific hyperplanes is such that different hyperplane competitions have conflicting winners [14]; however, since *delta coding* changes the way hyperspace is defined with respect to the objective function at each iteration, the same partitions of hyperspace will not exist from one *delta iteration* to the next.

The second problem that *delta coding* could perhaps resolve is the recombination of *lethals*. *Lethals* refer to poor offspring that result from the recombination of two good, but dissimilar parents. For example, the bit strings 00000 and 11111 may represent local optima, while points in between (e.g. 00111) would result in very poor solutions. Therefore, recombination between the dissimilar parents would generally produce worse solutions providing inconsistent hyperplane feedback [13]. *Delta coding* may help resolve such difficulties by redefining new hyperplane mappings in N-space and redefining the hypercube with respect to only one of the competing solutions.

ACKNOWLEDGEMENTS

This research was supported in part by NSF grant IRI-9010546 and in part by a grant from the Colorado Institute of Artificial Intelligence (CIAI). CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado.

References

[1] Ackley, D. (1987) *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers.

[2] Booker, L. (1987) "Improving Search in Genetic Algorithms," In, *Genetic Algorithms and Simulated Annealing.* L. Davis, ed. Pitman/Morgan Kaufmann.

[3] Baker, J., Greffenstette, J., (1989) "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism." *Proc. Third International Conf. on Genetic Algorithms.* Morgan Kaufmann.

[4] Eshelman, L., (1991) "The CHC Adaptive Search Algorithm: How to have Safe Search When Engaging in Nontraditional Genetic Recombination." *Foundations of Genetic Algorithms.* G. Rawlins, ed. Morgan Kaufmann.

[5] Goldberg, D., (1989) "Sizing Populations for Serial and Parallel Genetic Algorithms." *Proc. Third International Conf. on Genetic Algorithms.* Morgan Kaufmann.

[6] Nickerson, B., Fitzhorn, P., Koch, S., Charney, M., (1991) "A Method for Near Optimal Computational Superimposition of 2D Digital Facial Photographs and 3D Cranial Surface Meshes." *J. Forensic Sciences.* 2:480-500.

[7] Krishnakumar, K., (1989) "Micro-Genetic Algorithms for Stationary and Non-stationary Function Optimization." *SPIE's Intelligent Control and Adaptive Systems Conf.*, Paper # 1196-32.

[8] Mathias, K., (1991) "Delta Coding Strategies for Genetic Algorithms." *Thesis.*, Department of Computer Science, Colorado State University, Fort Collins, CO.

[9] Schraudolph, N., Belew, R., (1990) "Dynamic Parameter Encoding for Genetic Algorithms." *CSE Technical Report* #CS 90-175.

[10] Shaefer, C., (1987) "The ARGOT Strategy: Adaptive Representative Genetic Optimizer Technique." *Genetic Algorithms and their Applications: Proc of the Second International Conf.*

[11] Starkweather, T., Whitley, D., Mathias, K., (1990) "Optimization Using Distributed Genetic Algorithms." *Parallel Problem Solving from Nature.* Springer/Verlag.

[12] Whitley, D. and Starkweather, T., (1990) "*GENITOR II*: A Distributed Genetic Algorithm." *J. Experimental and Theoretical Artificial Intelligence.* 2:189-214.

[13] Whitley, D., Starkweather, T., and Bogart, C., (1990) "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity." *Parallel Computing.* 14:347-361.

[14] Whitley, D., (1991) "Fundamental Principles of Deception in Genetic Search." *Foundation of Genetic Algorithms.* G. Rawlins, ed. Morgan Kaufmann.

[15] Whitley, D., Hanson, T., (1989) "Optimizing Neural Networks Using Faster, More Accurate Genetic Search." *Proc. Third International Conf. on Genetic Algorithms.* Morgan Kaufmann.