Delta Coding

23-10-2023

Introduction

- What is Delta Coding?
  - Definition
  - Why Delta coding?
  - Algorithm
  - Advantages of Delta coding
- Performance comparisons
  - CHC, ESGA, GENITOR and RMHC
- Gray Coding (vs Binary coding)
  - Advantages
  - Performance difference
- Oscillation
- Overhead

Definition

- Delta coding is an iterative genetic search strategy that dynamically changes the representation of the search space in an attempt to exploit different problem respresentations.

Why Delta coding?

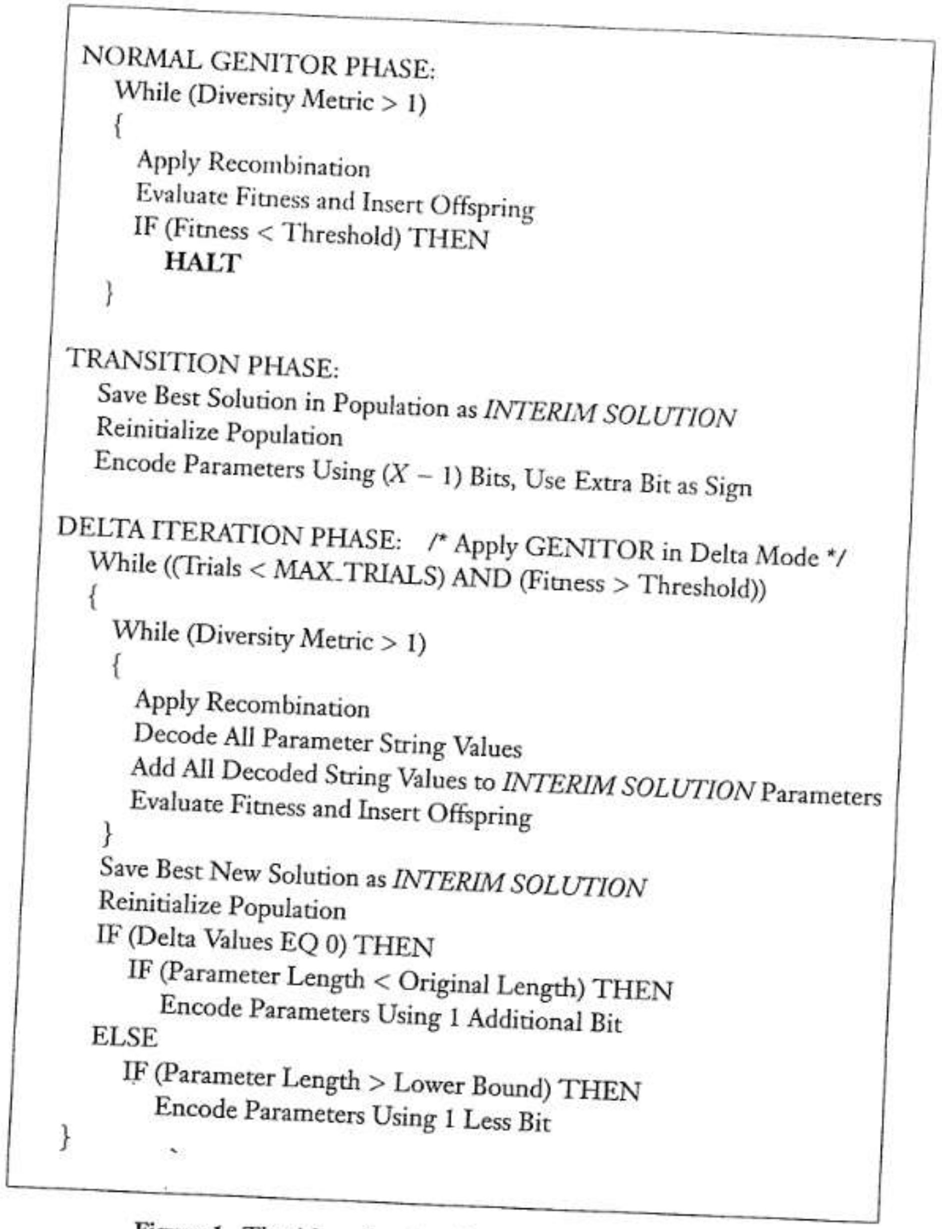
- Delta coding sustains search by reinitializing the population at each iteration of search
    - Avoids asymptotic performance when population becomes more homogeneous

Why Delta coding?

- Genetic Algorithms are very sensitive to the representation of the problem. The choice of it can determine whether a particular search method will succeed or fail.
  - Searching a problem space while dynamically changing the problem representation.
  - Leads to changes of the difficulty of a search problem.
  - Try to avoid the biases associated with one particular representation of the space
  - If some representations pose an ‘easier’ search problem or better performance for a genetic algorithm, it should be exploited.
    - E.g. 3 bit length vs 5 bit length or grey vs binary encoding

The Algorithm

- Three phases:
  - Initialization phase
  - Transition phase
  - Delta Iteration phase
- Diversity metric
- Reinitialize population
- Interim Solution
- Encoding parameters
- Decoding parameters
- Delta values (±[illegible])
- Changing amount of bits

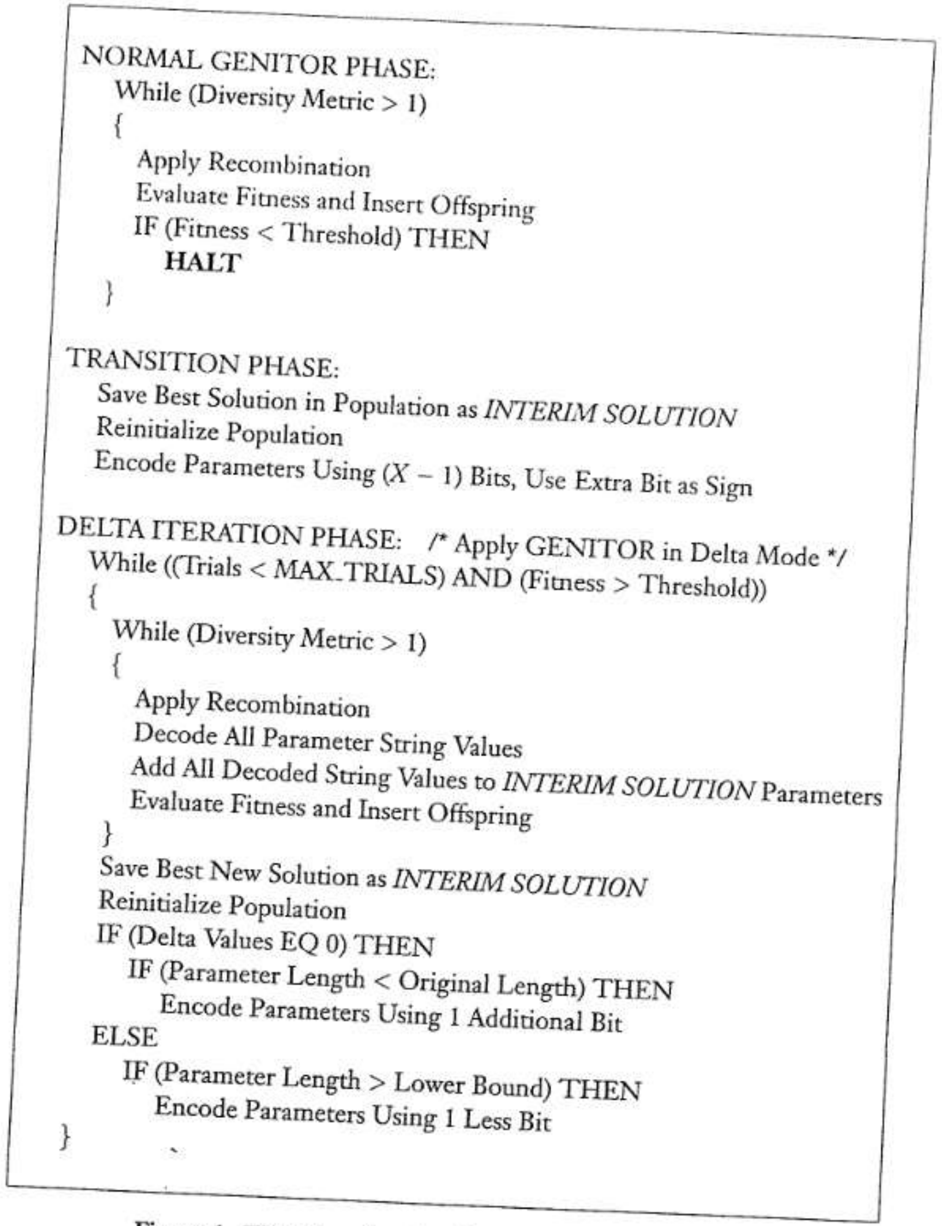


```
NORMAL GENITOR PHASE:
  While (Diversity Metric > 1)
  {
    Apply Recombination
    Evaluate Fitness and Insert Offspring
    IF (Fitness < Threshold) THEN
      HALT
  }

TRANSITION PHASE:
  Save Best Solution in Population as INTERIM SOLUTION
  Reinitialize Population
  Encode Parameters Using (X − 1) Bits, Use Extra Bit as Sign

DELTA ITERATION PHASE:   /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
    While (Diversity Metric > 1)
    {
      Apply Recombination
      Decode All Parameter String Values
      Add All Decoded String Values to INTERIM SOLUTION Parameters
      Evaluate Fitness and Insert Offspring
    }
    Save Best New Solution as INTERIM SOLUTION
    Reinitialize Population
    IF (Delta Values EQ 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 (assuming minimization).

Normal Genitor Phase

- First initial phase
  - Binary problem encoding
  - GENITOR engine
- Diversity metric
  - Comparing the best and worst strings in the current *persisting* population
    - Persisting is defined as the best N-1 strings in the population size of N
  - If the two strings are identical or vary by a single bit in the least significant position, search is suspended
  - Best solution is saved as **Interim Solution**

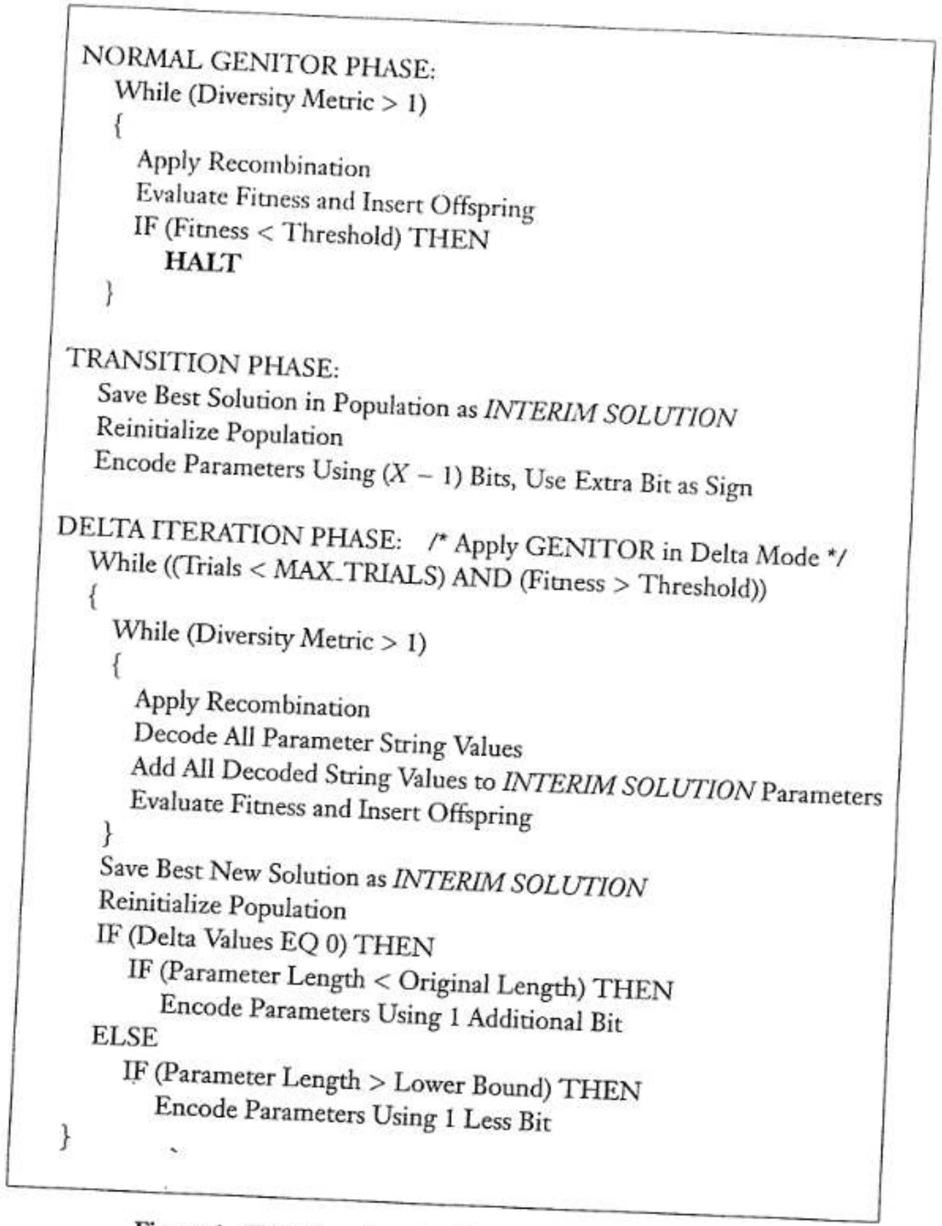


```
NORMAL GENITOR PHASE:
  While (Diversity Metric > 1)
  {
    Apply Recombination
    Evaluate Fitness and Insert Offspring
    IF (Fitness < Threshold) THEN
      HALT
  }

TRANSITION PHASE:
  Save Best Solution in Population as INTERIM SOLUTION
  Reinitialize Population
  Encode Parameters Using (X - 1) Bits, Use Extra Bit as Sign

DELTA ITERATION PHASE:    /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
    While (Diversity Metric > 1)
    {
      Apply Recombination
      Decode All Parameter String Values
      Add All Decoded String Values to INTERIM SOLUTION Parameters
      Evaluate Fitness and Insert Offspring
    }
    Save Best New Solution as INTERIM SOLUTION
    Reinitialize Population
    IF (Delta Values EQ 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 (assuming minimization).

Transition Phase

- Best solution is saved as **<u>Interim Solution</u>**
- Population is reinitialized randomly
- Encode Parameters
  - Using X-1 bits as normal bits and extra bit as <u>sign</u>
    - E.g. [1|1001]

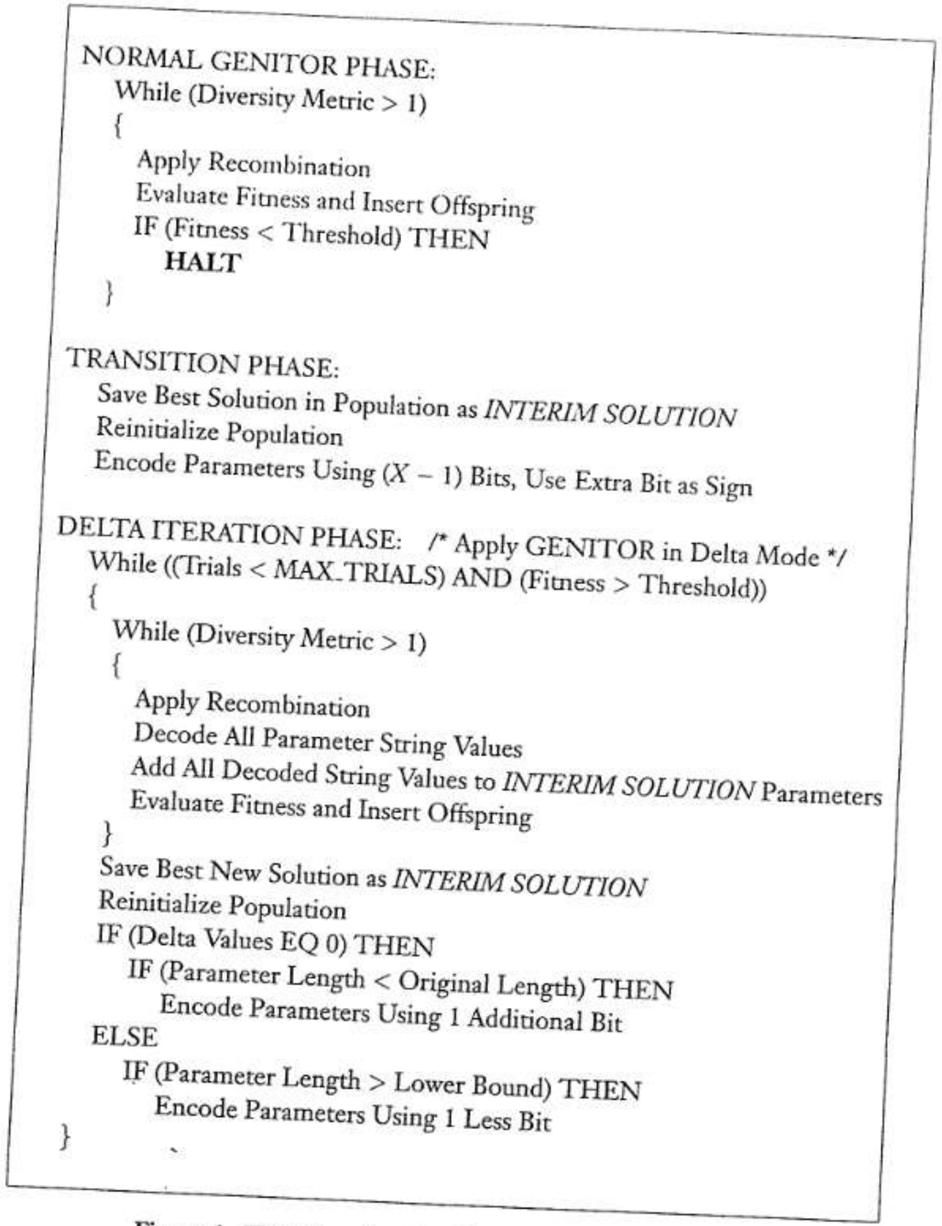


```
NORMAL GENITOR PHASE:
  While (Diversity Metric > 1)
  {
    Apply Recombination
    Evaluate Fitness and Insert Offspring
    IF (Fitness < Threshold) THEN
      HALT
  }

TRANSITION PHASE:
  Save Best Solution in Population as INTERIM SOLUTION
  Reinitialize Population
  Encode Parameters Using (X - 1) Bits, Use Extra Bit as Sign

DELTA ITERATION PHASE:    /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
    While (Diversity Metric > 1)
    {
      Apply Recombination
      Decode All Parameter String Values
      Add All Decoded String Values to INTERIM SOLUTION Parameters
      Evaluate Fitness and Insert Offspring
    }
    Save Best New Solution as INTERIM SOLUTION
    Reinitialize Population
    IF (Delta Values EQ 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 (assuming minimization).

Delta Iteration Phase

- Genitor is thus restarted
- Each parameter is assigned a <u>delta value (±□)</u> to the interim solution saved from previous iteration
    - Delta value is a measure of distance to the interim solution
- Searches a new hypercube with the interim solution to the origin
- The numerical range of each parameter is altered to allow the algorithm to search different subpartitions of the hyperspace
    - This is done by altering the number of bits used to represent each parameter



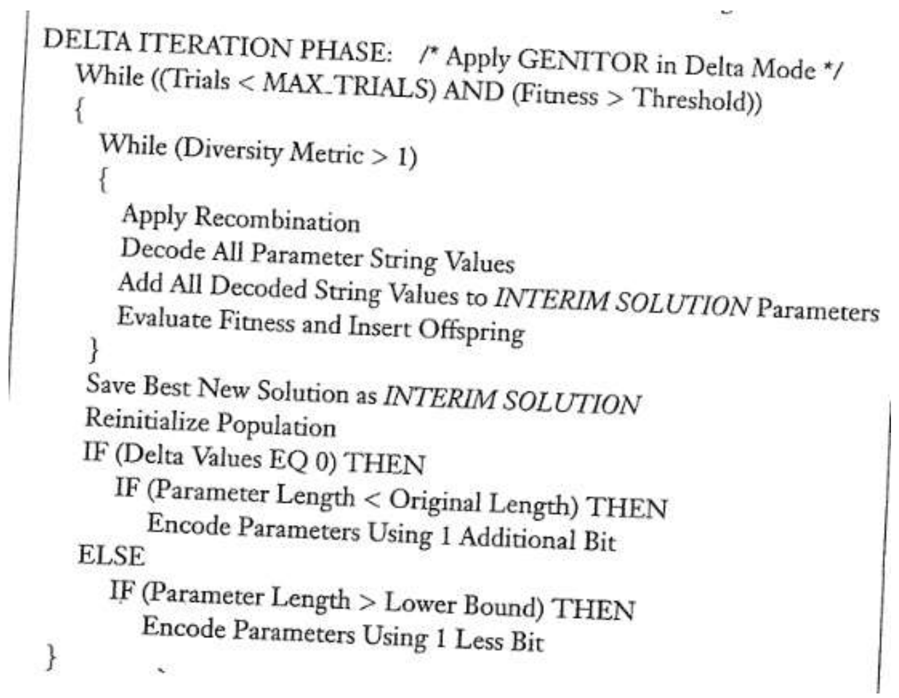
```
NORMAL GENITOR PHASE:
  While (Diversity Metric > 1)
  {
     Apply Recombination
     Evaluate Fitness and Insert Offspring
     IF (Fitness < Threshold) THEN
        HALT
  }

TRANSITION PHASE:
  Save Best Solution in Population as INTERIM SOLUTION
  Reinitialize Population
  Encode Parameters Using (X − 1) Bits, Use Extra Bit as Sign

DELTA ITERATION PHASE:   /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
     While (Diversity Metric > 1)
     {
        Apply Recombination
        Decode All Parameter String Values
        Add All Decoded String Values to INTERIM SOLUTION Parameters
        Evaluate Fitness and Insert Offspring
     }
     Save Best New Solution as INTERIM SOLUTION
     Reinitialize Population
     IF (Delta Values EQ 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 (assuming minimization).

Delta Iteration Phase

Decoding Parameters

- 3-bit string example with [000] as interim solution
- If sign bit is 0:
  - Add delta value to interim solution
    - E.g. 001 receives delta value 1: 010 = 2, and 011 = 3
- Is sign bit is 1:
  - Complement all other bits and add result
    - E.g. 100 will be 111 and delta value is equal to -3
- So Binary parameter 7 is adjacent to parameter 0 in this case
  - Only mapping is changed, not the number of bits



```
DELTA ITERATION PHASE:    /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
    While (Diversity Metric > 1)
    {
      Apply Recombination
      Decode All Parameter String Values
      Add All Decoded String Values to INTERIM SOLUTION Parameters
      Evaluate Fitness and Insert Offspring
    }
    Save Best New Solution as INTERIM SOLUTION
    Reinitialize Population
    IF (Delta Values EQ 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
}
```

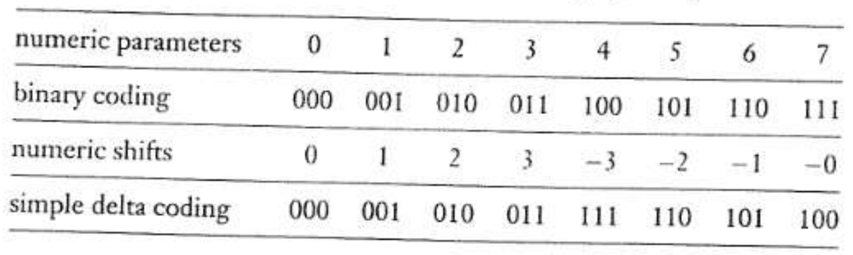


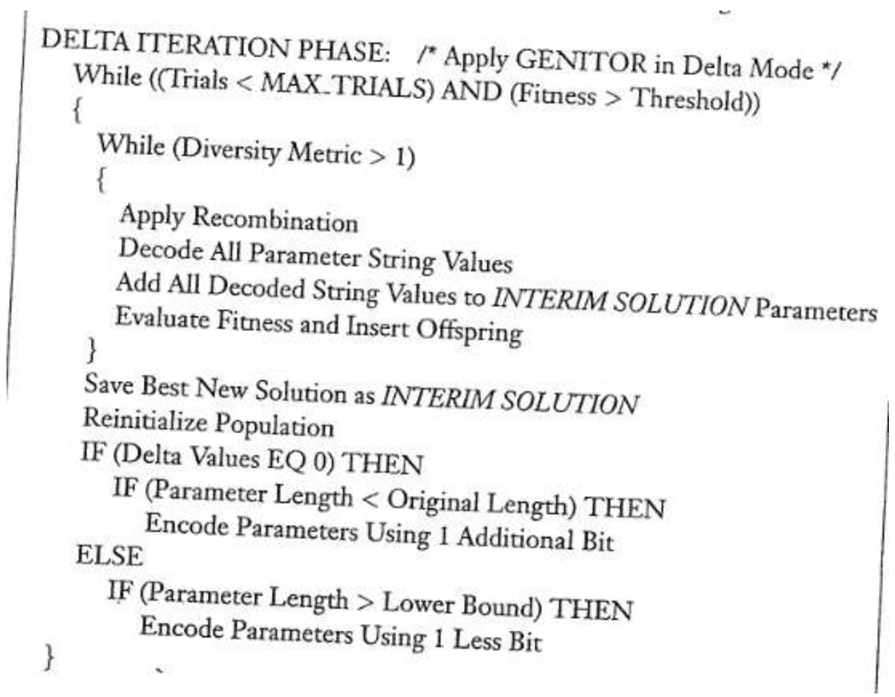
Table 1. Delta coding numeric shift (remapping) example.

numeric parameters	0	1	2	3	4	5	6	7
binary coding	000	001	010	011	100	101	110	111
numeric shifts	0	1	2	3	−3	−2	−1	−0
simple delta coding	000	001	010	011	111	110	101	100

Delta Iteration Phase

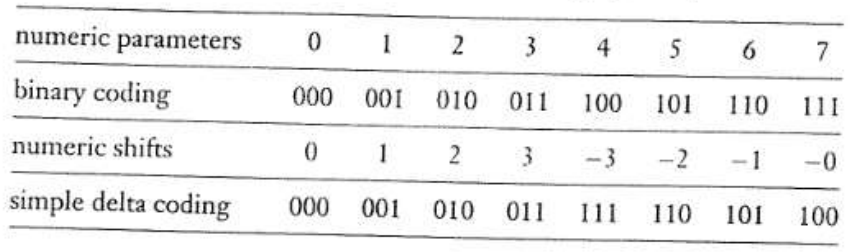
<u>Bit alternation</u>

- If the best new solution is not different from the previous interim solution (Delta values = 0)
  - And if parameter length is lower then original length
    - 1 additional bit
- If the best new solution is *<u>different</u>* from the previous interim solution
  - Higher then lower bound length (= to prevent search becoming too small)
    - 1 less bit



```
DELTA ITERATION PHASE:   /* Apply GENITOR in Delta Mode */
  While ((Trials < MAX_TRIALS) AND (Fitness > Threshold))
  {
    While (Diversity Metric > 1)
    {
      Apply Recombination
      Decode All Parameter String Values
      Add All Decoded String Values to INTERIM SOLUTION Parameters
      Evaluate Fitness and Insert Offspring
    }
    Save Best New Solution as INTERIM SOLUTION
    Reinitialize Population
    IF (Delta Values EQ 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
}
```

Table 1. Delta coding numeric shift (remapping) example.



numeric parameters	0	1	2	3	4	5	6	7
binary coding	000	001	010	011	100	101	110	111
numeric shifts	0	1	2	3	−3	−2	−1	−0
simple delta coding	000	001	010	011	111	110	101	100

Delta Iteration Phase

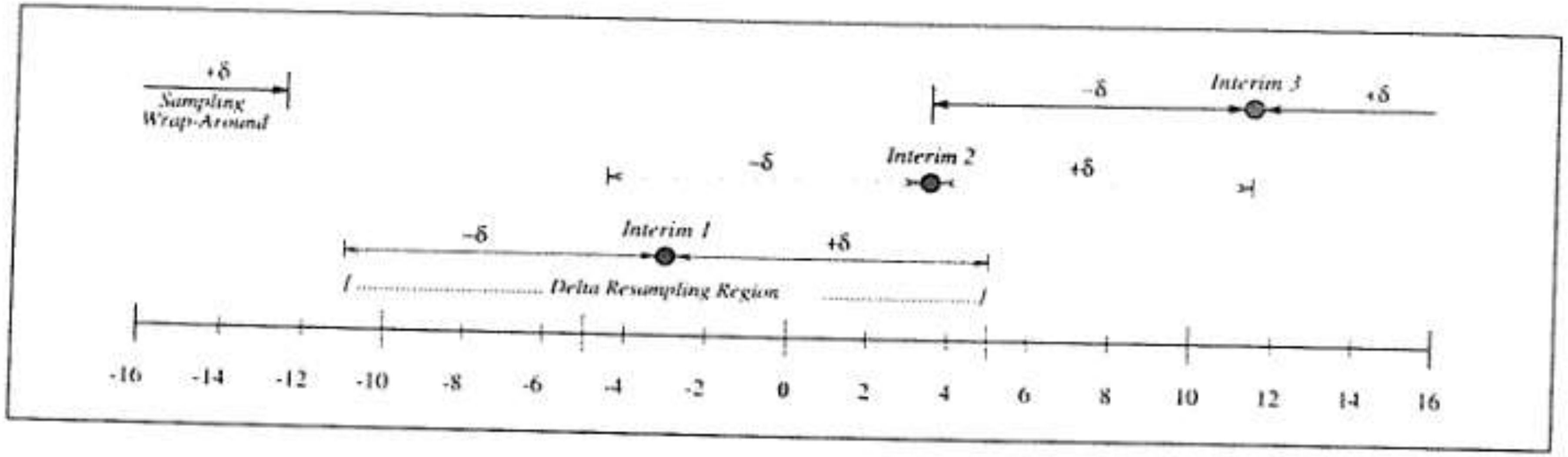


Figure 2. Points sampled in a one-dimensional numeric space using delta coding.

If the interim solution is close to boundary of the search space, then sampling will be discontinued and resumed to the opposite boundary of the space

Advantages of Delta Coding

- It is conceptually simple to understand and implement
- Does not rely on disruptive mechanisms to sustain diversity during genetic search
  - Searches until the diversity of current population is exhausted
  - Saves the best known solution parameters at that point and reinitializes population which provides a new and diverse population to resume the search resulting in heterogeneous behaviour.
- Adjusting the size of the hypercube on the solution space are a function of the current population diversity and previous interim solution
  - No complex mechanisms
  - It could also allow shorter populations in reduced hypercubes.

Advantages of Remapping Hyperspaces

- Not to locate ‘easiest’ mapping of the function, but an easier mapping than those explore earlier
  - No attempt is made to preserve previous relationships in Hamming space.

Advantages of Remapping Hyperspaces

The strings in Figure 3a are organized according to Hamming Distance with respect to 0000 and then ascending nummeric value; **global optimum at 1111 with deceptive attractor 0000**

The strings in Figure 3b are organized same as 3a. **Global optimum 1111 with two local optima: 1000 and 1111, so no 0000**

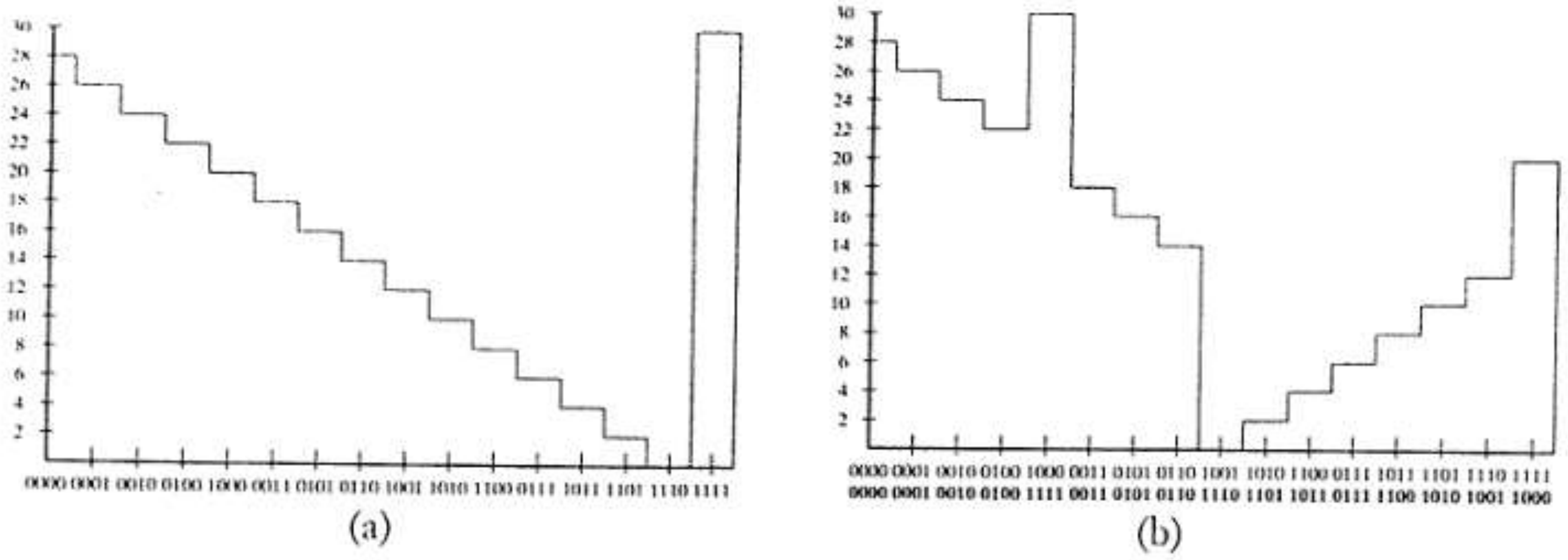


Figure 3. Remapping search space with delta coding; interim solution 0000. Figure (b) includes the delta encoding (row 1) and the original encoding (row 2).

Advantages of Subpartition Sample

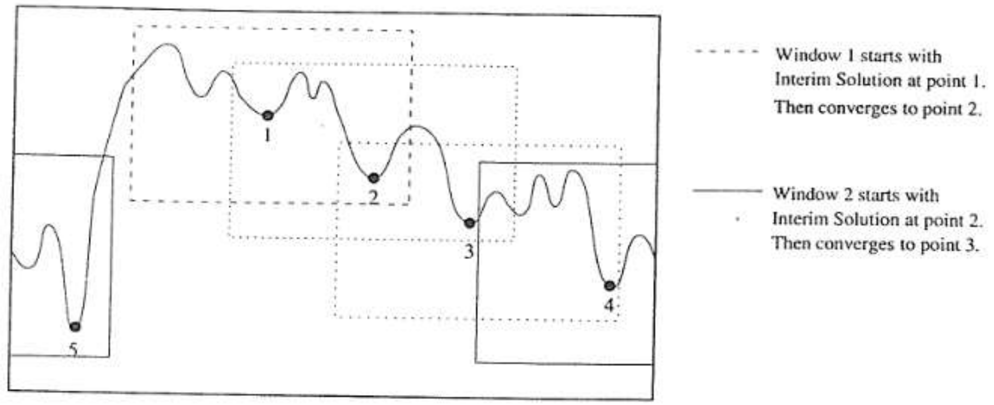

1
2
3
4
5
Window 1 starts with Interim Solution at point 1. Then converges to point 2.
Window 2 starts with Interim Solution at point 2. Then converges to point 3.

Figure 4. 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.

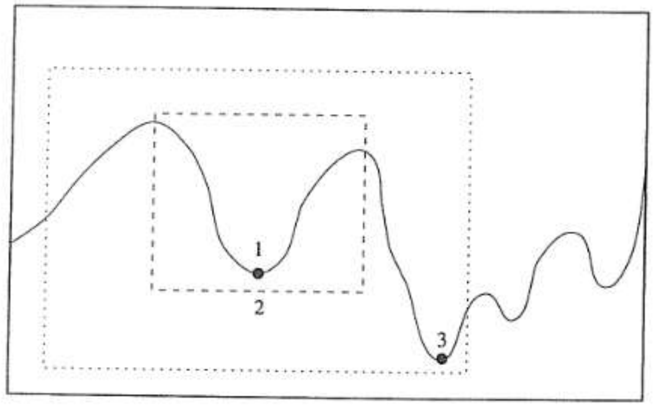


Figure 5. Expanding the delta coding search window after converging to the same interim solution on consecutive iterations using a reduced search space string representation.

Performance Comparisons of Delta Coding

- CHC
- Elitist Simple Genetic Algorithm (ESGA)
- GENITOR
- Random mutation hill-climbing (RMHC)

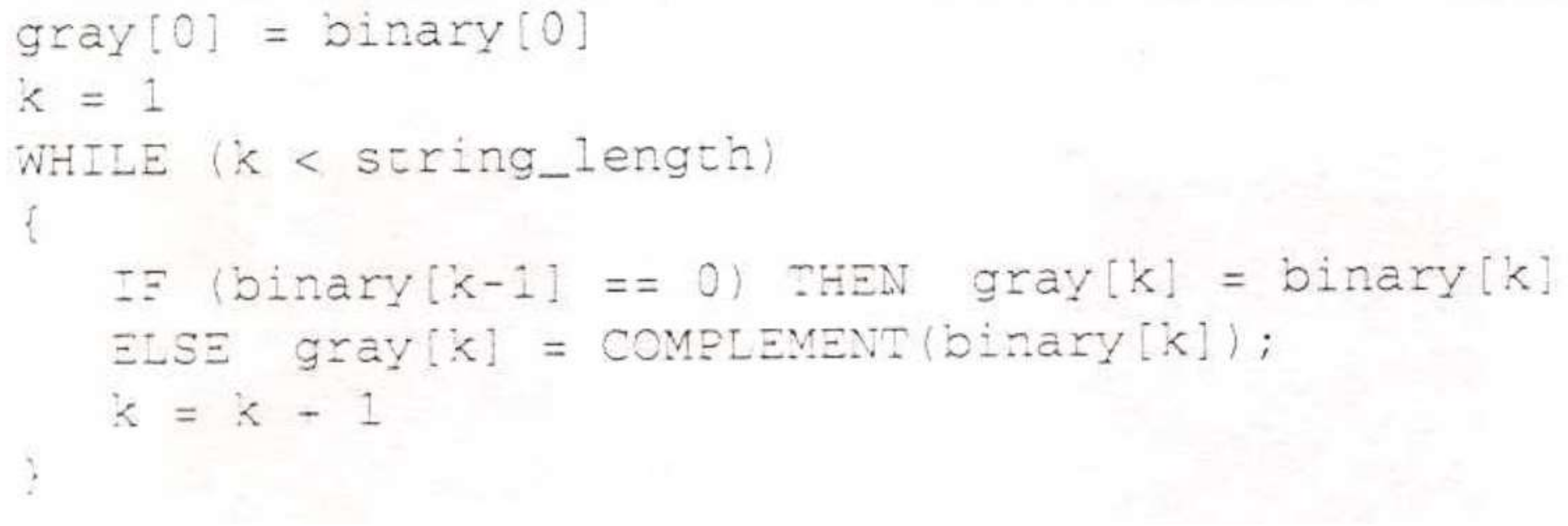
- F1 - F5: De Jong
- F6: Rastrigin
- F7: Schwefel
- F8: Griwank

Empirical Test Design

- Try to tune parameters of each algorithm such that the performance is as much as possible improved
- Best performance measure: Greatest number of runs locating the optimal solution while expending the least amount of work possible (E.g. the smallest number of recombinations)

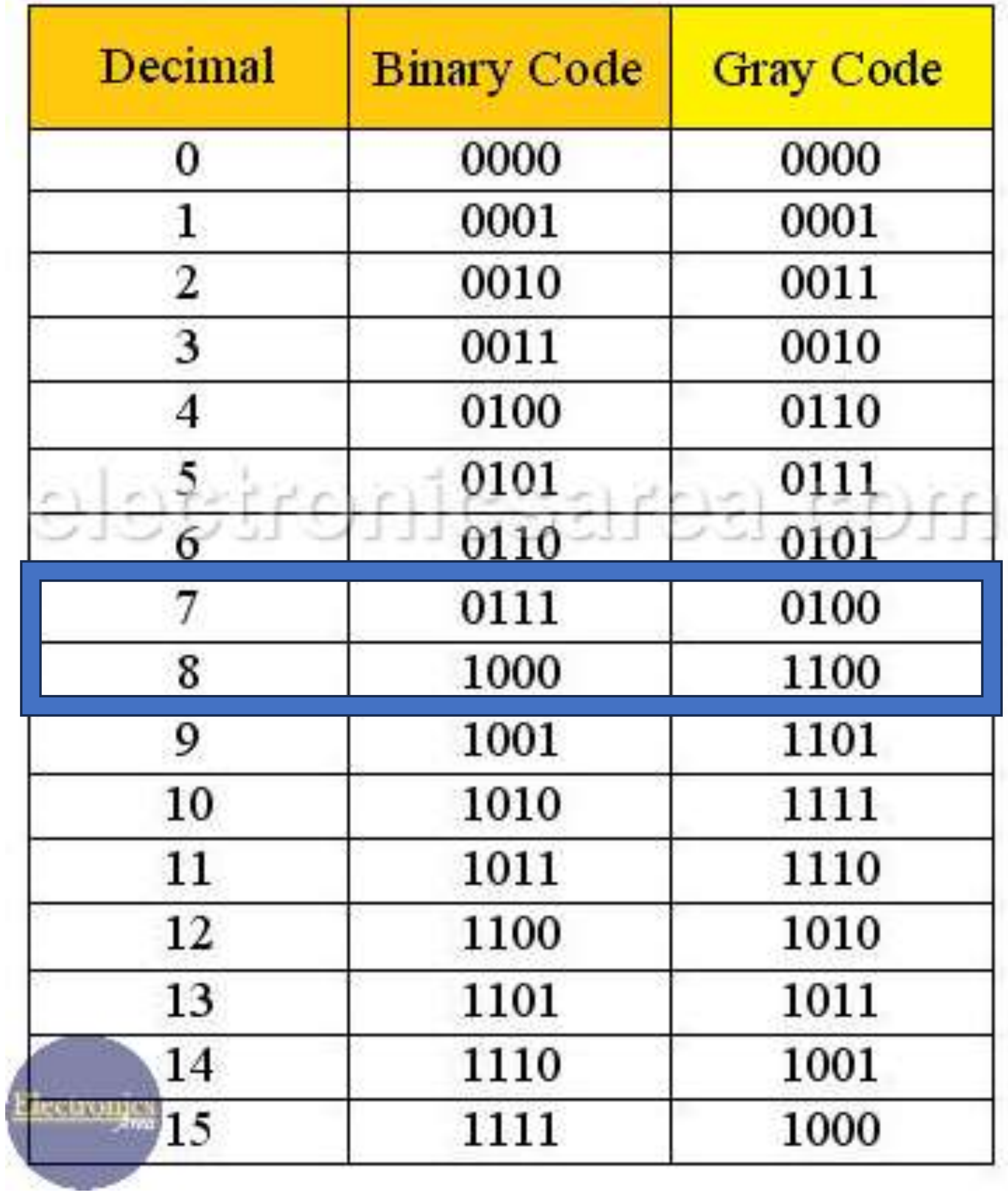
Gray coding

- Gray coding is a general method for transforming one binary mapping into another such that the resulting consectutive binary representations differ in a single bit position
- Binary Reflected Gray Code



```
gray[0] = binary[0]
k = 1
WHILE (k < string_length)
{
   IF (binary[k-1] == 0) THEN  gray[k] = binary[k]
   ELSE  gray[k] = COMPLEMENT(binary[k]);
   k = k + 1
}
```

Gray coding



Decimal	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Results (with Gray Coding)

- **Percent Solved**: Percentage optimal solutions found
- **Average Trials**: Average number of recombinations over sucessful runs
- **Average best and maximum trials** only when optimal is not 100% found
- **Average best**: Average fitness of the best individual in the population for all 30 runs after the maximum of recombinations have been executed

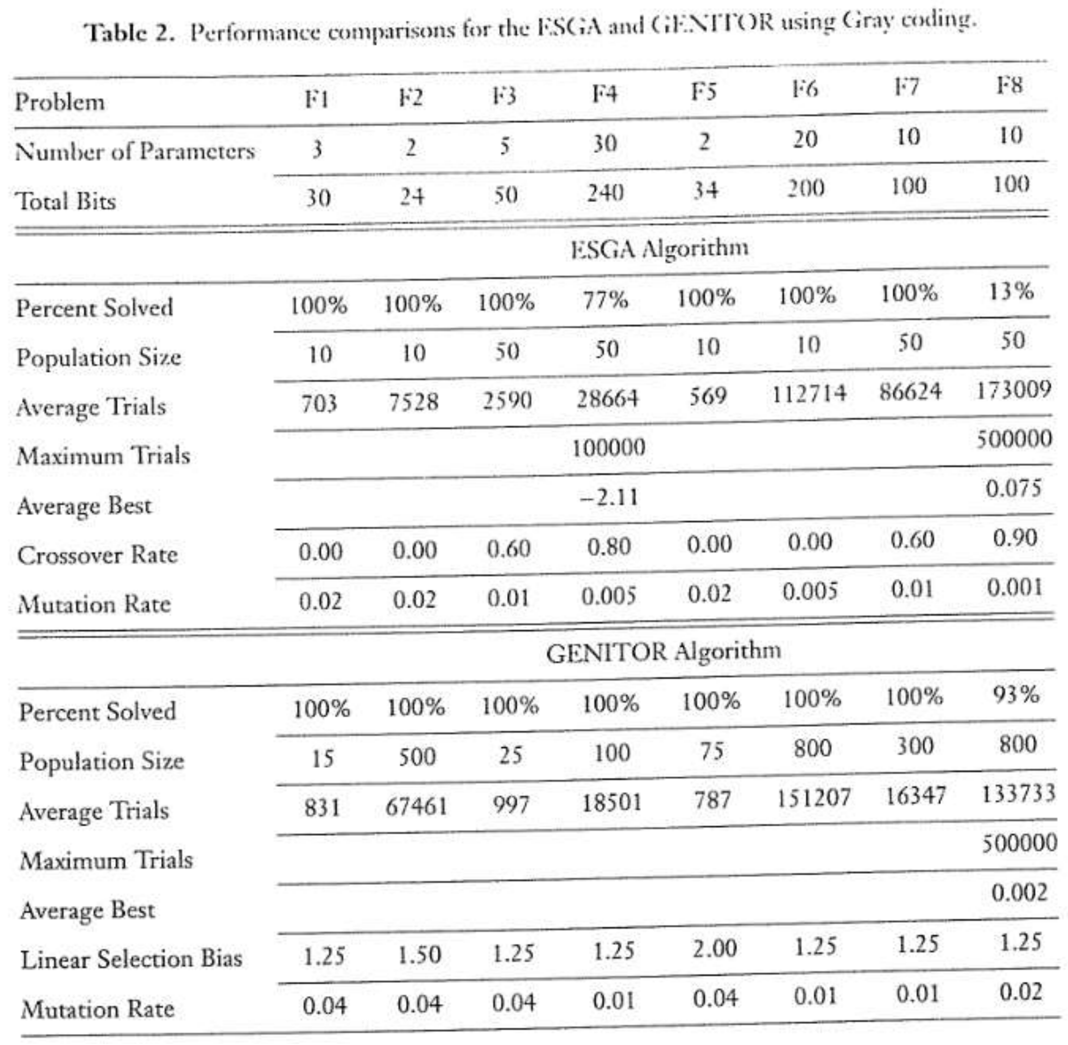


Table 2. Performance comparisons for the ESGA and GENITOR using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Number of Parameters	3	2	5	30	2	20	10	10
Total Bits	30	24	50	240	34	200	100	100
				ESGA Algorithm				
Percent Solved	100%	100%	100%	77%	100%	100%	100%	13%
Population Size	10	10	50	50	10	10	50	50
Average Trials	703	7528	2590	28664	569	112714	86624	173009
Maximum Trials				100000				500000
Average Best				−2.11				0.075
Crossover Rate	0.00	0.00	0.60	0.80	0.00	0.00	0.60	0.90
Mutation Rate	0.02	0.02	0.01	0.005	0.02	0.005	0.01	0.001
				GENITOR Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	93%
Population Size	15	500	25	100	75	800	300	800
Average Trials	831	67461	997	18501	787	151207	16347	133733
Maximum Trials								500000
Average Best								0.002
Linear Selection Bias	1.25	1.50	1.25	1.25	2.00	1.25	1.25	1.25
Mutation Rate	0.04	0.04	0.04	0.01	0.04	0.01	0.01	0.02

Results (with Gray Coding)

- ESGA best performed when no crossover is used
  - When no crossover is used, it always performs better then GENITOR
- GENITOR finds the optimal solution consistently (besides F8)
- GENITOR performed best when mutation was added

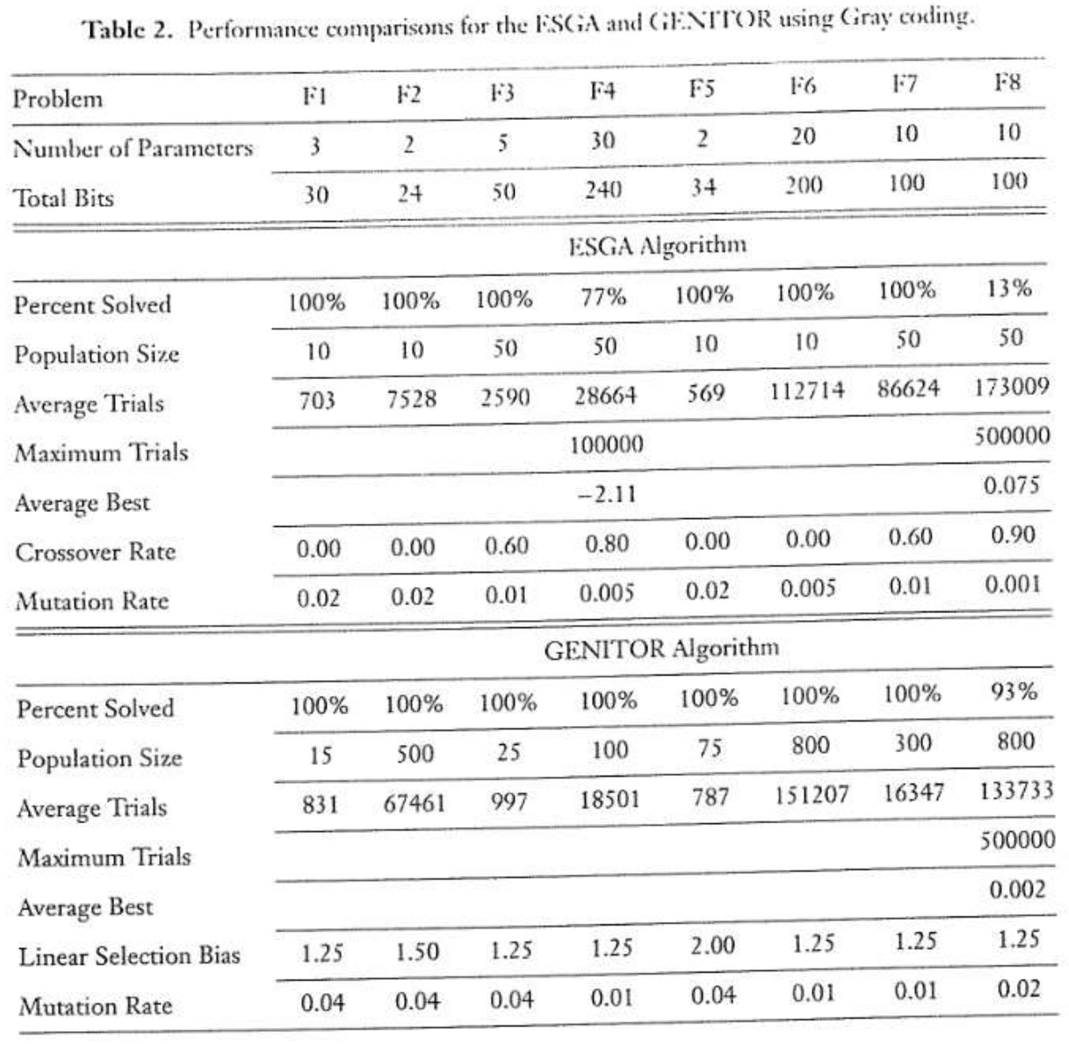


Table 2. Performance comparisons for the ESGA and GENITOR using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Number of Parameters	3	2	5	30	2	20	10	10
Total Bits	30	24	50	240	34	200	100	100
ESGA Algorithm								
Percent Solved	100%	100%	100%	77%	100%	100%	100%	13%
Population Size	10	10	50	50	10	10	50	50
Average Trials	703	7528	2590	28664	569	112714	86624	173009
Maximum Trials				100000				500000
Average Best				−2.11				0.075
Crossover Rate	0.00	0.00	0.60	0.80	0.00	0.00	0.60	0.90
Mutation Rate	0.02	0.02	0.01	0.005	0.02	0.005	0.01	0.001
GENITOR Algorithm								
Percent Solved	100%	100%	100%	100%	100%	100%	100%	93%
Population Size	15	500	25	100	75	800	300	800
Average Trials	831	67461	997	18501	787	151207	16347	133733
Maximum Trials								500000
Average Best								0.002
Linear Selection Bias	1.25	1.50	1.25	1.25	2.00	1.25	1.25	1.25
Mutation Rate	0.04	0.04	0.04	0.01	0.04	0.01	0.01	0.02

Results (with Gray Coding)

- Suprisingly, RMHC consisently finds the optimal solution for six out of eight test functions, performing better then ESGA and GENITOR on four of those six cases
- Initial run for Delta coding used Grey coding, the subsequent runs used signed binary coding with a lower bound of 4-bits
- CHC solved F7 and F8 using fewer recombinations than any other algorithm and only algorithm besides delta coding to solve consistently
- Delta coding performed best or second best on all functions besides F6 and F7
    - Delta Coding performed especially well on F2, F4 and F8 which proved to be very difficult for other functions

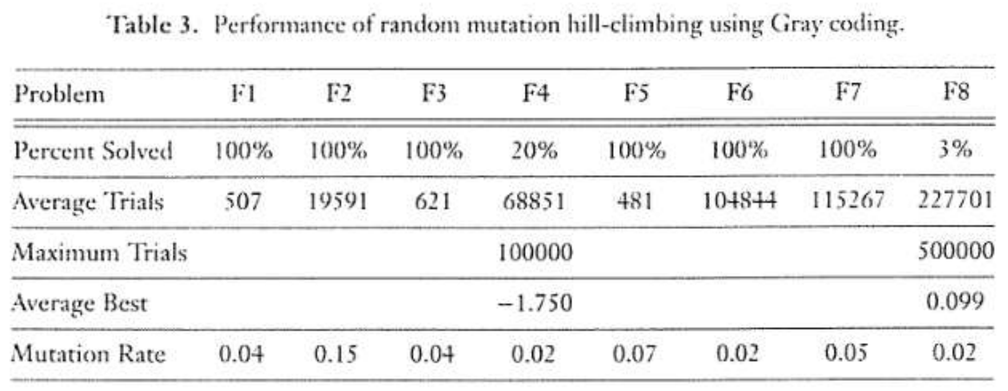


Table 3. Performance of random mutation hill-climbing using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Percent Solved	100%	100%	100%	20%	100%	100%	100%	3%
Average Trials	507	19591	621	68851	481	104844	115267	227701
Maximum Trials				100000				500000
Average Best				−1.750				0.099
Mutation Rate	0.04	0.15	0.04	0.02	0.07	0.02	0.05	0.02

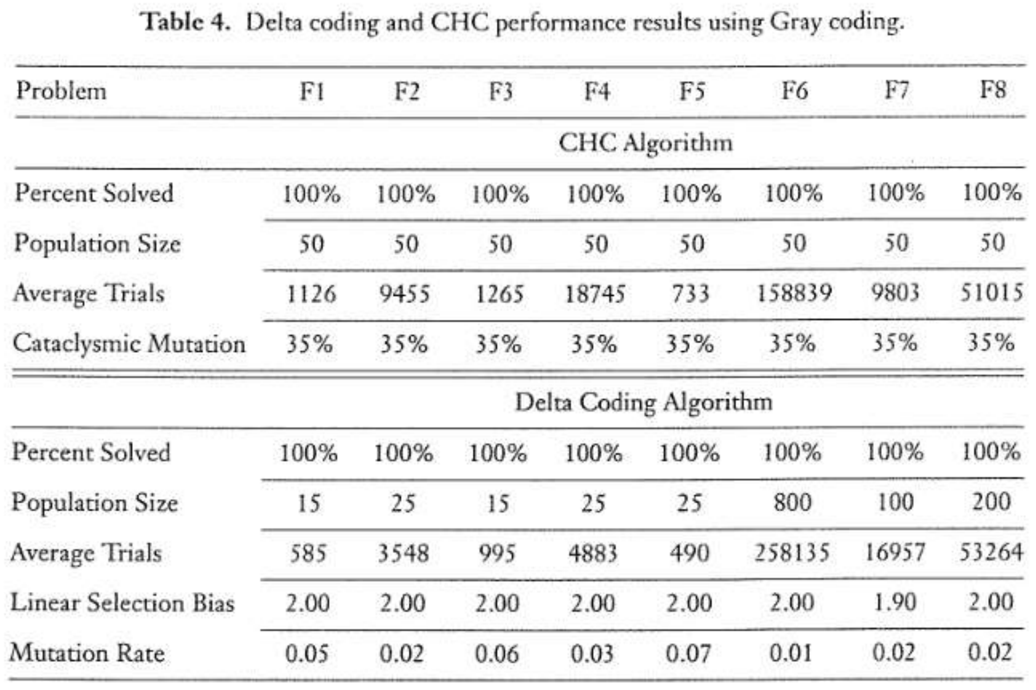


Table 4. Delta coding and CHC performance results using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
				CHC Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
Population Size	50	50	50	50	50	50	50	50
Average Trials	1126	9455	1265	18745	733	158839	9803	51015
Cataclysmic Mutation	35%	35%	35%	35%	35%	35%	35%	35%
				Delta Coding Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
Population Size	15	25	15	25	25	800	100	200
Average Trials	585	3548	995	4883	490	258135	16957	53264
Linear Selection Bias	2.00	2.00	2.00	2.00	2.00	2.00	1.90	2.00
Mutation Rate	0.05	0.02	0.06	0.03	0.07	0.01	0.02	0.02

Results (with Gray Coding)

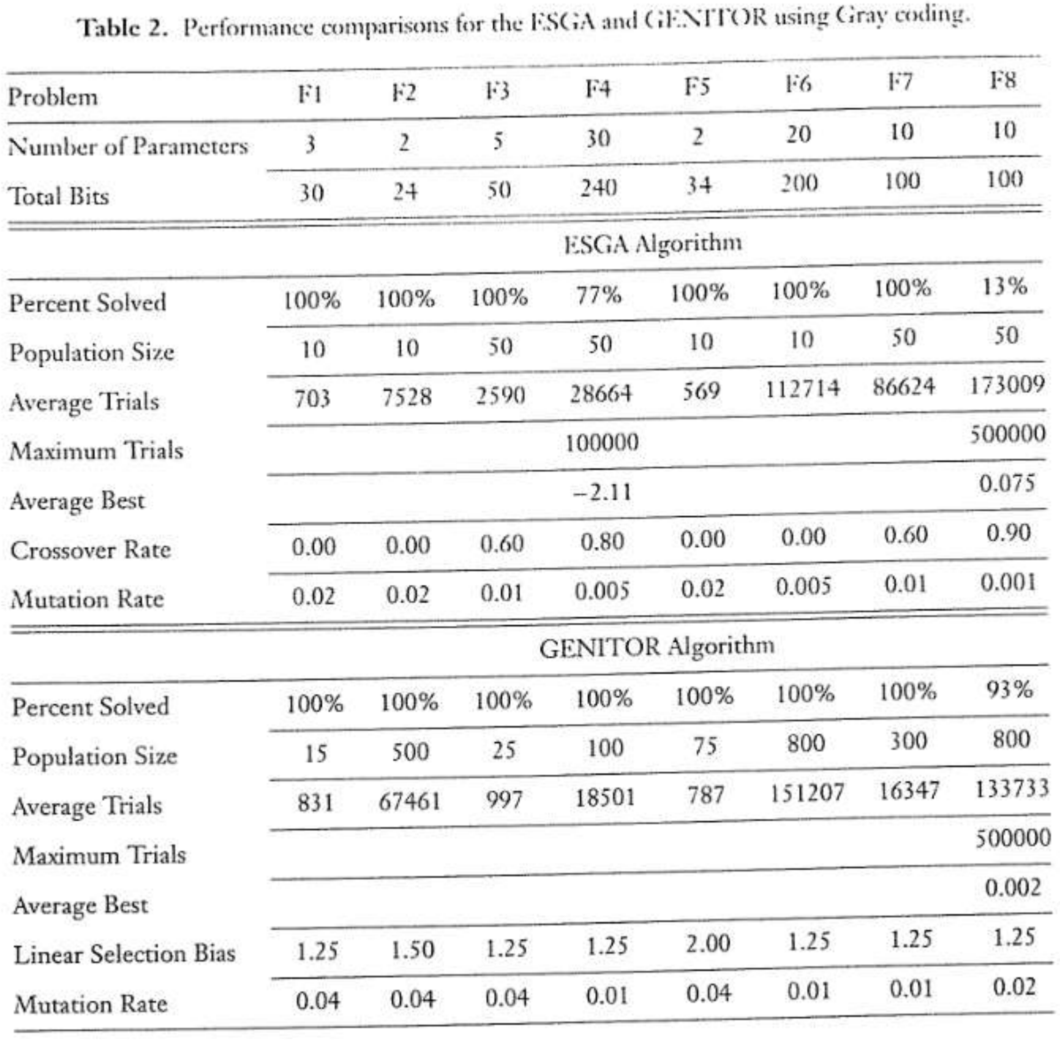


Table 2. Performance comparisons for the ESGA and GENITOR using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Number of Parameters	3	2	5	30	2	20	10	10
Total Bits	30	24	50	240	34	200	100	100
				ESGA Algorithm				
Percent Solved	100%	100%	100%	77%	100%	100%	100%	13%
Population Size	10	10	50	50	10	10	50	50
Average Trials	703	7528	2590	28664	569	112714	86624	173009
Maximum Trials				100000				500000
Average Best				−2.11				0.075
Crossover Rate	0.00	0.00	0.60	0.80	0.00	0.00	0.60	0.90
Mutation Rate	0.02	0.02	0.01	0.005	0.02	0.005	0.01	0.001
				GENITOR Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	93%
Population Size	15	500	25	100	75	800	300	800
Average Trials	831	67461	997	18501	787	151207	16347	133733
Maximum Trials								500000
Average Best								0.002
Linear Selection Bias	1.25	1.50	1.25	1.25	2.00	1.25	1.25	1.25
Mutation Rate	0.04	0.04	0.04	0.01	0.04	0.01	0.01	0.02

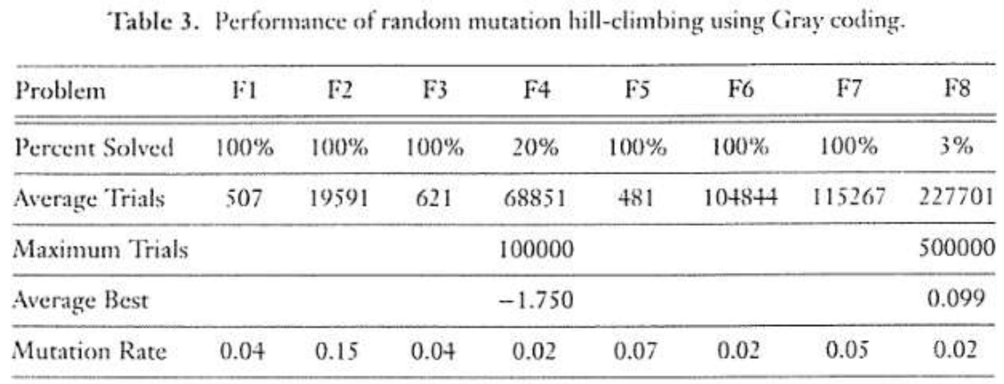


Table 3. Performance of random mutation hill-climbing using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
Percent Solved	100%	100%	100%	20%	100%	100%	100%	3%
Average Trials	507	19591	621	68851	481	104844	115267	227701
Maximum Trials				100000				500000
Average Best				−1.750				0.099
Mutation Rate	0.04	0.15	0.04	0.02	0.07	0.02	0.05	0.02

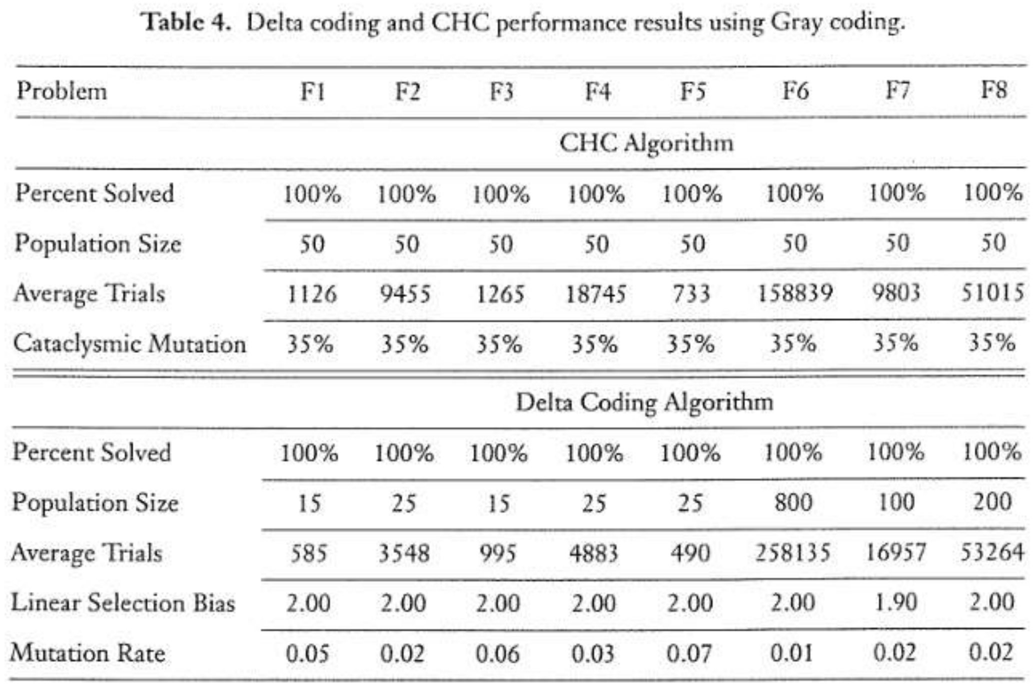


Table 4. Delta coding and CHC performance results using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
				CHC Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
Population Size	50	50	50	50	50	50	50	50
Average Trials	1126	9455	1265	18745	733	158839	9803	51015
Cataclysmic Mutation	35%	35%	35%	35%	35%	35%	35%	35%
				Delta Coding Algorithm				
Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
Population Size	15	25	15	25	25	800	100	200
Average Trials	585	3548	995	4883	490	258135	16957	53264
Linear Selection Bias	2.00	2.00	2.00	2.00	2.00	2.00	1.90	2.00
Mutation Rate	0.05	0.02	0.06	0.03	0.07	0.01	0.02	0.02

Should Grey Coding be used?

- Gray coding not only eliminates Hamming Cliffs but also has the potential to significantly alter the number of local optima in the search space as well as the size of the basins of attraction.

- Gray coding also induces a new set of hyperplane relationships, thus changing the schema competitions during genetic search.

Hamming cliffs

- A Hamming cliff occurs when two consecutive numbers have complementary binary representations
  - E.g. binary numbers 7 and 8 (1000 and 0111)
  - Consequently, may not locate optimal solution
  - Property: **two local optima in binary space will be in the same attraction basin in Grey coded space**

Number of local optima

- Number of local optima for F6 until F8 functions for 10-bit strings
- F6 and F7
    - Fewer Local optima for Grey coding
    - Number of local optima grow the formula: l^d
    - **As dimensionality increases, Grey coding solves a simpler function mapping**
- F8
    - approximately equal
    - More rapid then l^d

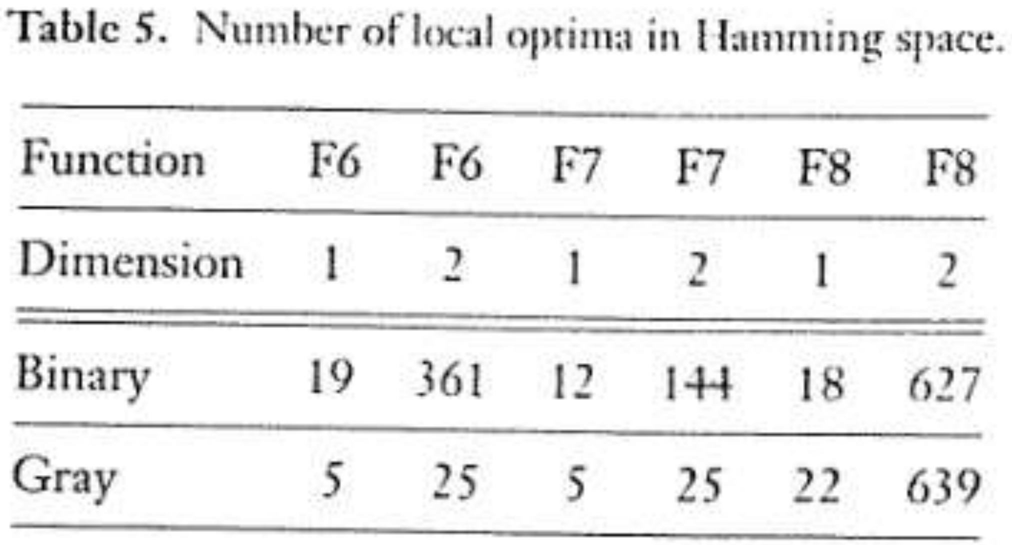


Table 5. Number of local optima in Hamming space.

Function	F6	F6	F7	F7	F8	F8
Dimension	1	2	1	2	1	2
Binary	19	361	12	144	18	627
Gray	5	25	5	25	22	639

Number of local optima

- Number of local optima for F6 until F8 functions for 10-bit strings
- F6 and F7
  - Fewer Local optima for Grey coding
  - Number of local optima grow the formula: l^d
  - **As dimensionality increases, Grey coding solves a simpler function mapping**
- F8
  - approximately equal
  - More rapid then l^d

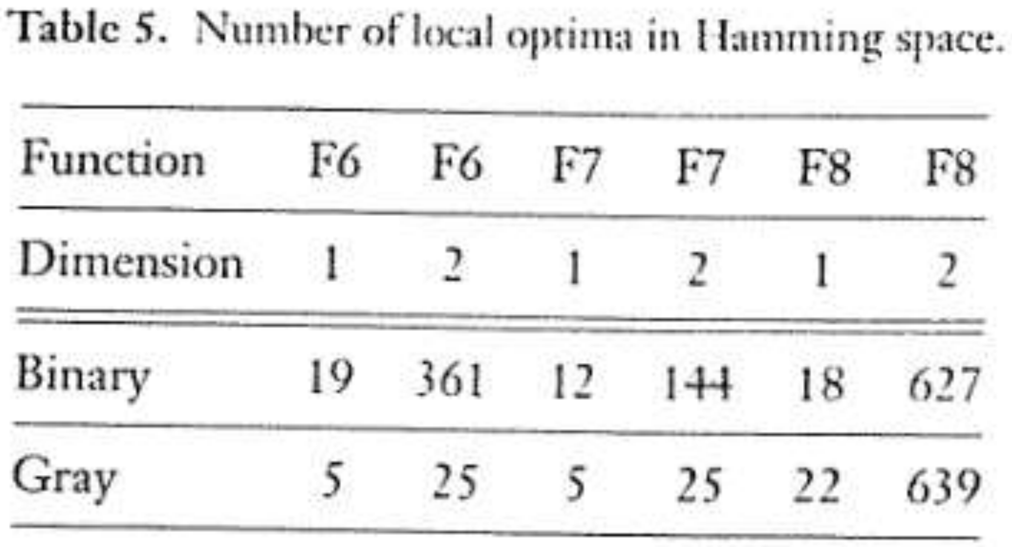


Table 5. Number of local optima in Hamming space.

Function	F6	F6	F7	F7	F8	F8
Dimension	1	2	1	2	1	2
Binary	19	361	12	144	18	627
Gray	5	25	5	25	22	639

Example

- Binary
  - Local minima: 0001, 0010 and 1110
  - Global minimum: 1000

- Gray Code:
  - Local Minima: 0011 and 1001
  - Global Minimum: 1100

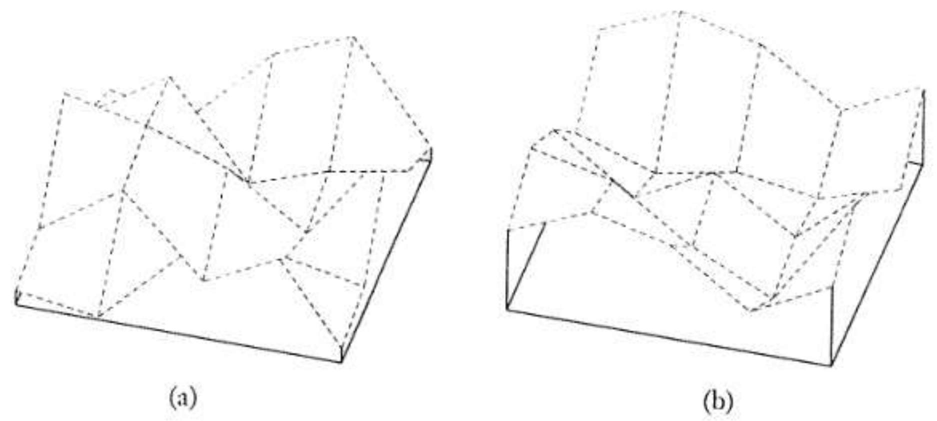


Figure 6. Three-dimensional view of binary and Gray coded Hamming space for a four-bit version of the Griewank function. The Gray coded space (b) is noticeably simpler than the binary space (a).

```
1111 – 1110 – 1010 – 1011 – 1111
 |      |      |      |      |
1101 – 1100 – 1000 – 1001 – 1101
 |      |      |      |      |
0101 – 0100 – 0000 – 0001 – 0101
 |      |      |      |      |
0111 – 0110 – 0010 – 0011 – 0111
 |      |      |      |      |
1111 – 1110 – 1010 – 1011 – 1111
```

Results (Binary vs Grey)

- Steepest Descent Search
  - Start at random point in Hamming Space
  - Evaluates l of its neighbors (l = length of string)
  - Algorithm moves to neighbor with smallest fitness value if it is equal or less then current point
  - Repeated until a better solution cannot be founded
  - If optimum is not reached, a new random starting point will be generated

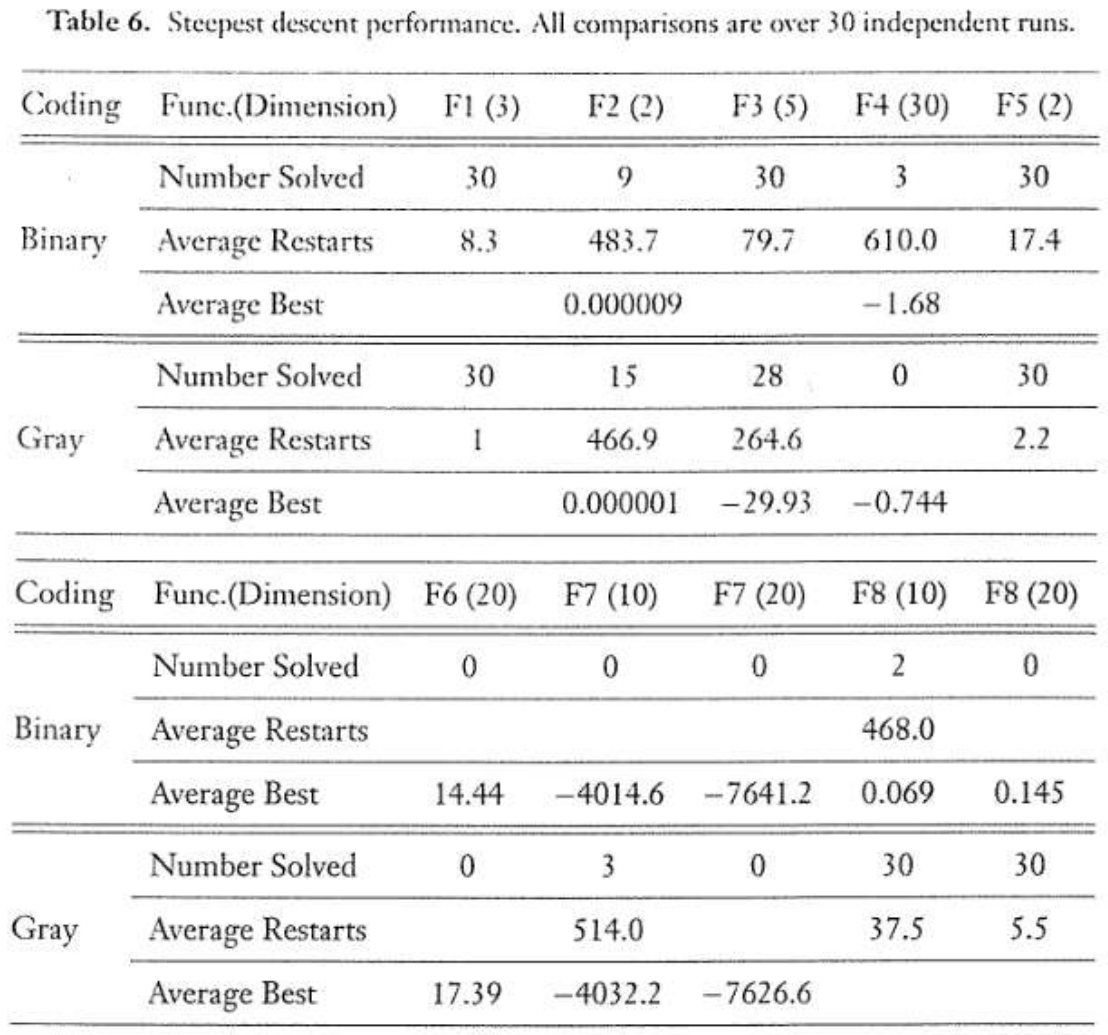


Table 6. Steepest descent performance. All comparisons are over 30 independent runs.

Coding	Func.(Dimension)	F1 (3)	F2 (2)	F3 (5)	F4 (30)	F5 (2)
	Number Solved	30	9	30	3	30
Binary	Average Restarts	8.3	483.7	79.7	610.0	17.4
	Average Best		0.000009		−1.68	
	Number Solved	30	15	28	0	30
Gray	Average Restarts	1	466.9	264.6		2.2
	Average Best		0.000001	−29.93	−0.744	

Coding	Func.(Dimension)	F6 (20)	F7 (10)	F7 (20)	F8 (10)	F8 (20)
	Number Solved	0	0	0	2	0
Binary	Average Restarts				468.0	
	Average Best	14.44	−4014.6	−7641.2	0.069	0.145
	Number Solved	0	3	0	30	30
Gray	Average Restarts		514.0		37.5	5.5
	Average Best	17.39	−4032.2	−7626.6		

Results (Binary vs Grey)

- **Grey Coding significantly enhances performance of Steepest Decent Hill for most of the functions**
  - More often
  - Fewer average trials
  - Better average solution

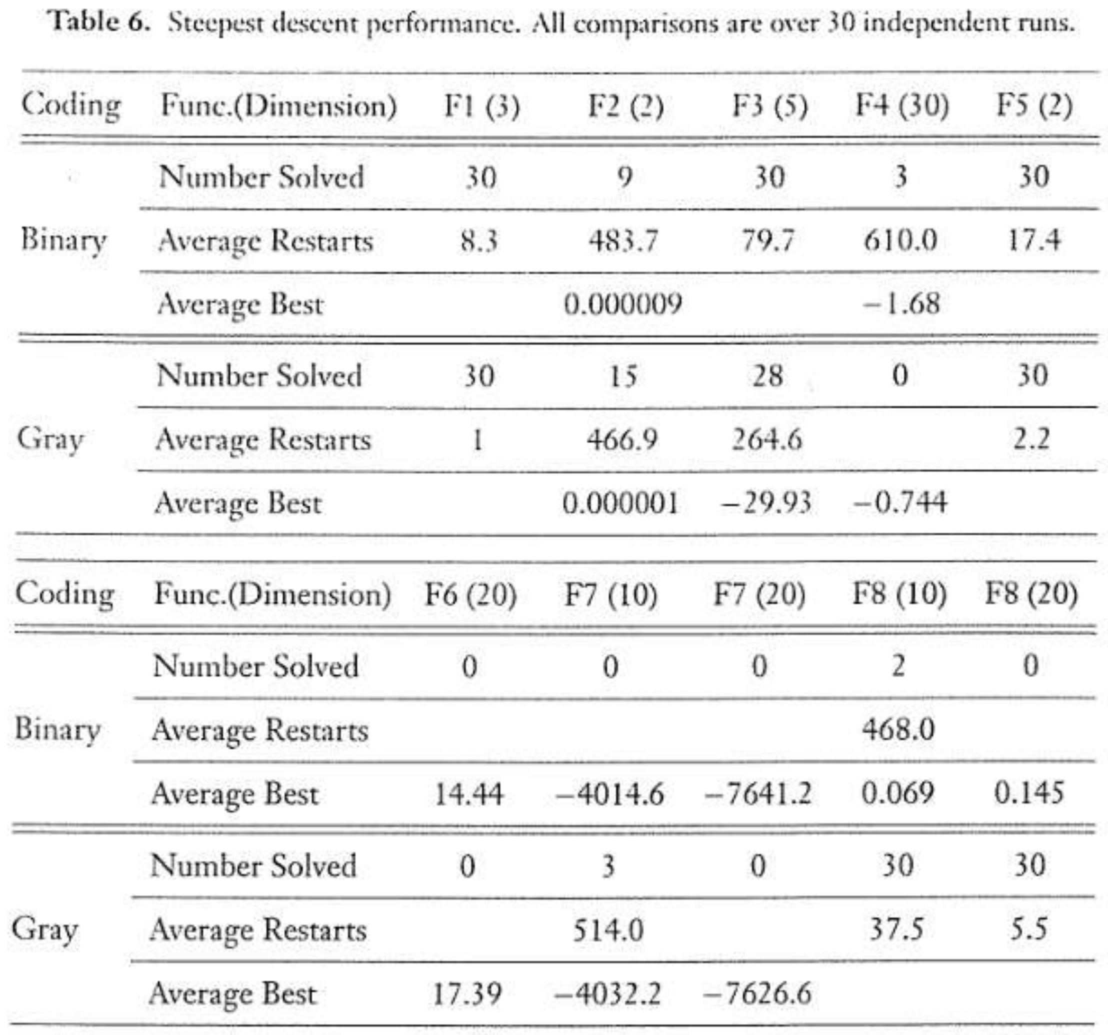


Table 6. Steepest descent performance. All comparisons are over 30 independent runs.

Coding	Func.(Dimension)	F1 (3)	F2 (2)	F3 (5)	F4 (30)	F5 (2)
Binary	Number Solved	30	9	30	3	30
	Average Restarts	8.3	483.7	79.7	610.0	17.4
	Average Best		0.000009		−1.68	
Gray	Number Solved	30	15	28	0	30
	Average Restarts	1	466.9	264.6		2.2
	Average Best		0.000001	−29.93	−0.744	

Coding	Func.(Dimension)	F6 (20)	F7 (10)	F7 (20)	F8 (10)	F8 (20)
Binary	Number Solved	0	0	0	2	0
	Average Restarts				468.0	
	Average Best	14.44	−4014.6	−7641.2	0.069	0.145
Gray	Number Solved	0	3	0	30	30
	Average Restarts		514.0		37.5	5.5
	Average Best	17.39	−4032.2	−7626.6		

Attraction basins

- Suprising for F8, based on the number of local optima difference between encodings
  - Proved be equally difficult
- The percentage of points in the attraction basin of the global optimum increases dramatically when grey coding is applied
- Global attraction basin in Grey space is 43% while 35% for Binary space
- The size of the attraction basin containing the global optimum increases relative to other basins of attraction as the dimentionality is increased

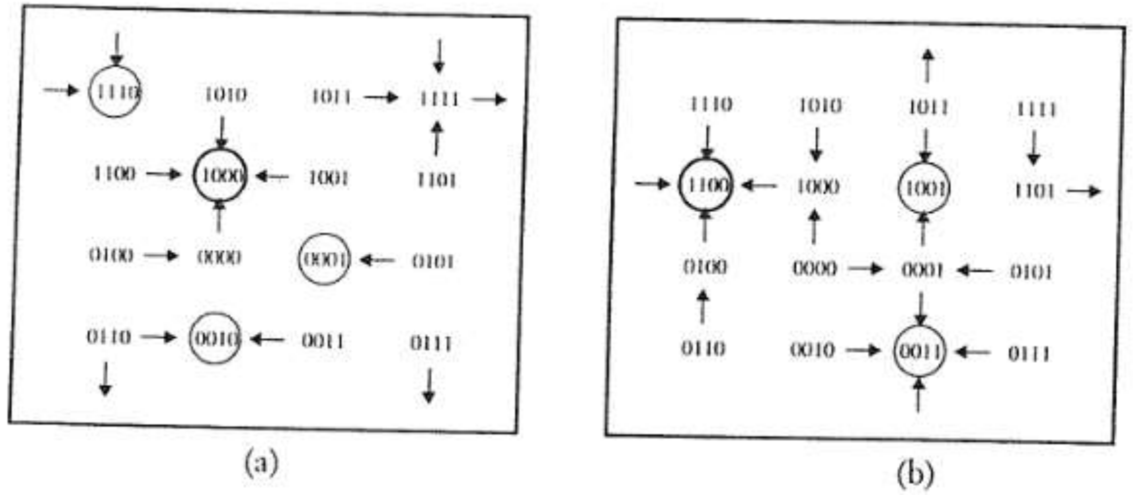


Figure 7. Binary and Gray coded Hamming space with respect to steepest ascent.

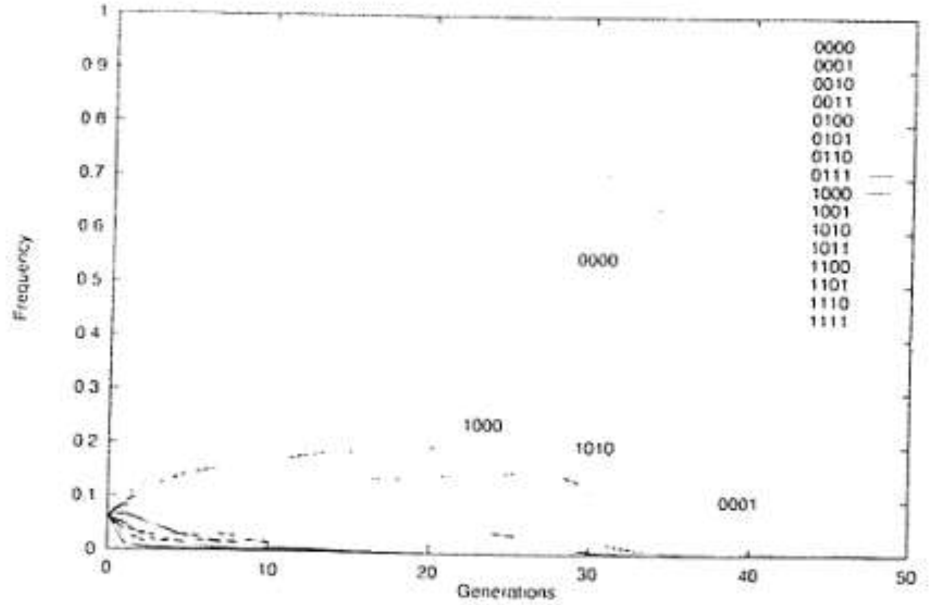
Should Grey Coding be used?

- Gray coding not only <u>eliminates Hamming Cliffs</u> but also has the potential to significantly <u>alter the number of local optima</u> in the search space as well as <u>the size of the basins of attraction</u>.

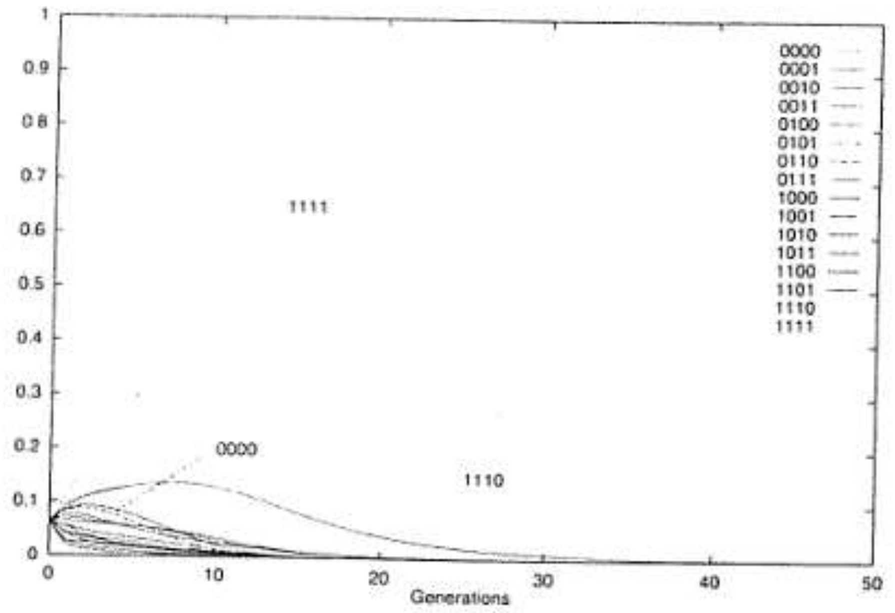
- Gray coding also induces a new set of hyperplane relationships, thus changing the schema competitions during genetic search.

Table 7. Four-bit function.

Binary	Gray	Fitness	Binary	Gray	Fitness
0000	0000	30	0100	0110	12
0001	0001	23	0101	0111	20
0010	0011	8	0110	0101	10
0011	0010	4	0111	0100	2
1000	1100	18	1100	1010	16
1001	1101	24	1101	1011	22
1010	1111	28	1110	1001	14
1011	1110	26	1111	1000	0



(a)



(b)

Results (without Grey Coding)

- RMHC solves only one of the test functions consistently
- CHC cannot solve F6 and F8 consistently
- Delta coding solves every test function consistently

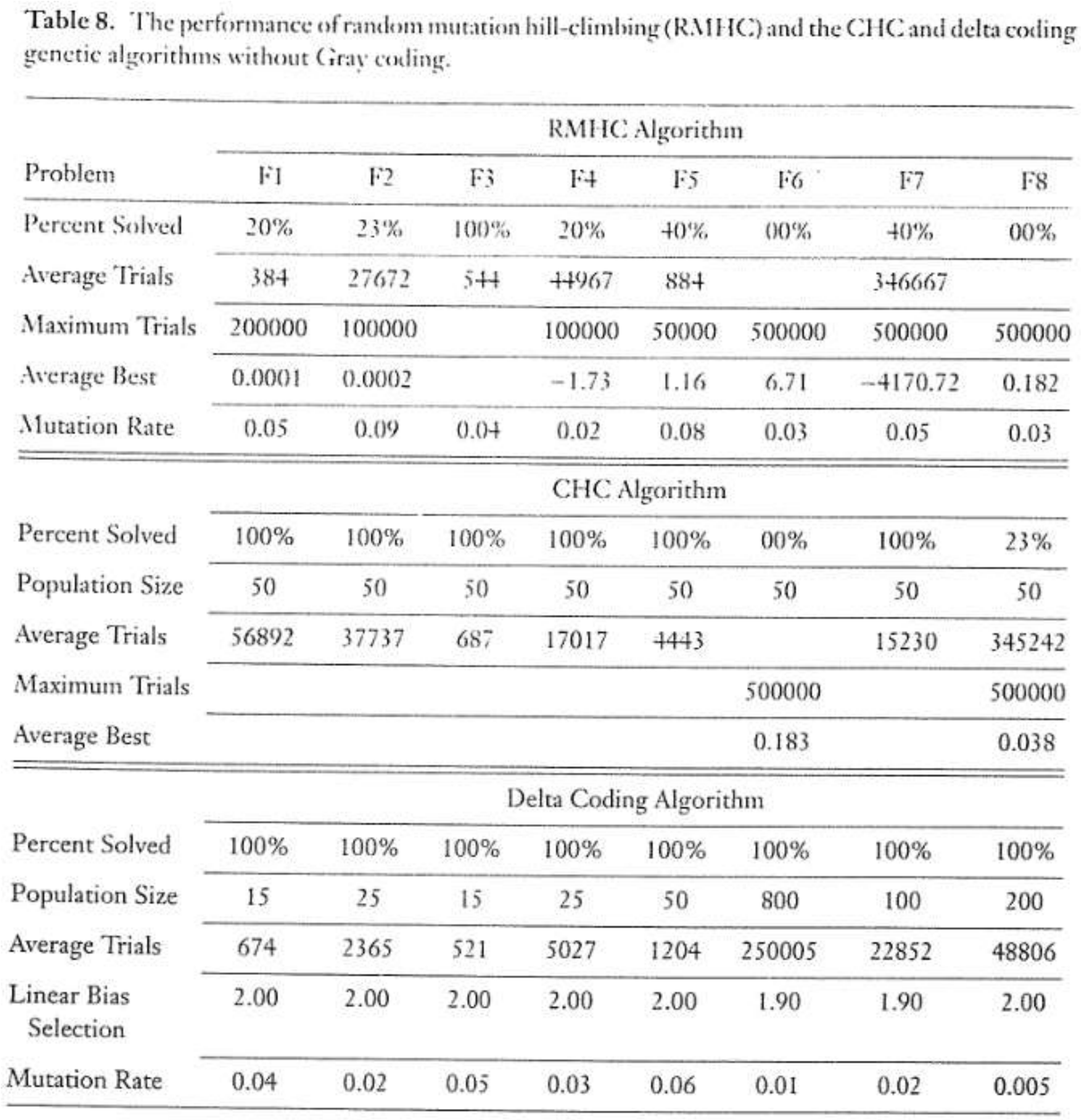


Table 8. The performance of random mutation hill-climbing (RMHC) and the CHC and delta coding genetic algorithms without Gray coding.

	RMHC Algorithm							
Problem	F1	F2	F3	F4	F5	F6	F7	F8
Percent Solved	20%	23%	100%	20%	40%	00%	40%	00%
Average Trials	384	27672	544	44967	884		346667	
Maximum Trials	200000	100000		100000	50000	500000	500000	500000
Average Best	0.0001	0.0002		−1.73	1.16	6.71	−4170.72	0.182
Mutation Rate	0.05	0.09	0.04	0.02	0.08	0.03	0.05	0.03
	CHC Algorithm							
Percent Solved	100%	100%	100%	100%	100%	00%	100%	23%
Population Size	50	50	50	50	50	50	50	50
Average Trials	56892	37737	687	17017	4443		15230	345242
Maximum Trials						500000		500000
Average Best						0.183		0.038
	Delta Coding Algorithm							
Percent Solved	100%	100%	100%	100%	100%	100%	100%	100%
Population Size	15	25	15	25	50	800	100	200
Average Trials	674	2365	521	5027	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	0.03	0.06	0.01	0.02	0.005

Oscillation in Delta Coding

- F6 and F7 are more challenging for Delta Coding
  - Oscillation condition

- Oscillation occures when the number of bits representing each parameter has been reduced to a lower limit such that the parameter representations cannot be reduced further
  - Converges to solution: A
  - In the next iteration: B
  - The solution is different so no increasement and cannot be reduced further
  - So next iteration: A

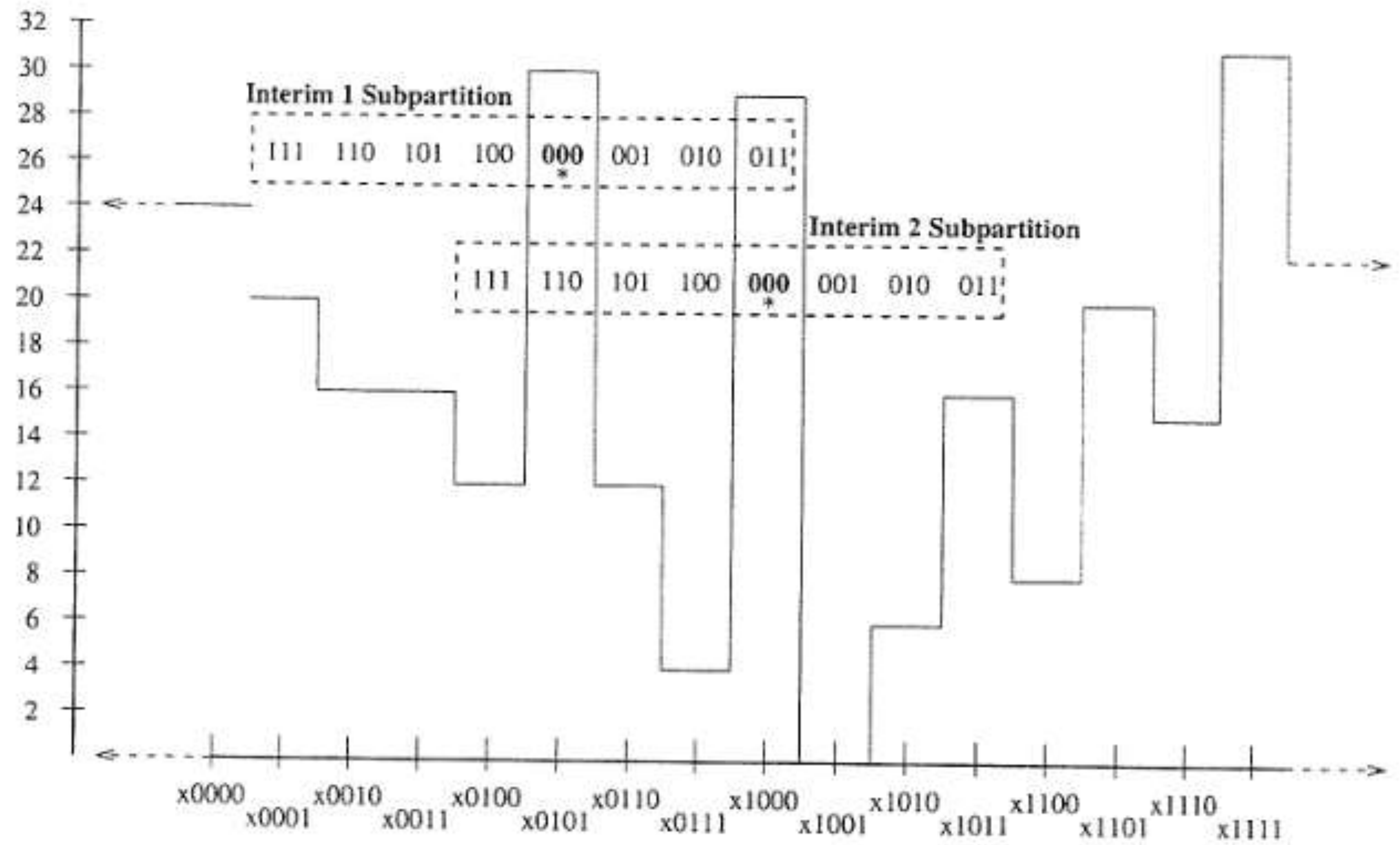


Figure 9. Oscillation problem search space with interim encodings. Note that the *x* prefacing each of the four-bit strings indicates that this subspace is a part of a larger search space.

How to deal with Oscillation?

- Save most recent interim solutions for several previous iterations
  - Oscillation can then be identified
    - Enlarge the search window by increasing number of bits
      - Single bit
      - Back to original length
- Test combinations of parameter values in most recent previous interim solutions
  - E.g. if there exists 4 function parameters, combine first and second parameter of one interim solution with third and fourth from another interim solution
    - Results in new set of input parameters for evaluation
- Seeding the population with strings that represented those parameters previous iterations ago with respect to the current interim solution
  - Search space is broadened again

Weakness Delta Coding

- Creates offspring which is worse then the current population
- Offspring are never inserted in the rank position of GENITOR serach mechanism
- Diversity does not change so diversity criterium will not be met
- Thus, reaches maximum number of trials
- Changing the maximum number of trials influenced the performance of Delta coding

Overhead of Delta Coding

- Used criteria overlook the relatively high Overhead in Delta Coding
- Two reasons:
  - Testing the Hamming distance between population members to check for population diversity to continue the search
  - The total reinitialization of the pupulation for each delta iteration
- Delta Coding is one of the most expensive algorithms
  - So Time measurement gives a better indication

Table 9. CPU seconds to solve test suite functions for random mutation hill-climbing (RMHC), CHC, and delta coding algorithms using Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
				RMHC Algorithm				
% Solved	100%	100%	100%	37%	100%	100%	100%	3%
Avg. Time	0.15	5.63	0.36	458.00	0.29	254.76	180.47	573.19
σ	0.06	3.77	0.02	146.85	0.21	123.96	79.46	43.02
Avg. Best				−1.92				0.099
Significance	> *both*		> *both*		> *chc*	> *δ coding*		
				CHC Algorithm				
% Solved	100%	100%	100%	100%	100%	100%	100%	100%
Avg. Time	0.57	1.91	0.76	36.45	0.46	284.44	11.51	50.49
σ	0.19	0.85	0.24	16.49	0.21	128.94	3.31	40.21
Significance		> *rmhc*		> *rmhc*		> *δ coding*	> *both*	> *rmhc*
				Delta Coding Algorithm				
% Solved	100%	100%	100%	100%	100%	100%	100%	100%
Avg. Time	0.31	1.40	0.82	12.45	0.34	804.29	25.75	65.64
σ	0.10	0.63	0.40	4.05	0.12	65.71	10.25	28.84
Significance	> *chc*	> *both*		> *both*	> *chc*		> *rmhc*	> *rmhc*

Table 10. CPU seconds to solve test suite functions for random mutation hill-climbing (RMHC), CHC, and delta coding algorithms *without* Gray coding.

Problem	F1	F2	F3	F4	F5	F6	F7	F8
				RMHC Algorithm				
% Solved	20%	30%	100%	27%	40%	0%	23%	0%
Avg. Time	138.51	107.48	0.30	295.18	17.46	1615.50	676.25	745.73
σ	70.38	50.54	0.18	84.63	14.26	7.19	106.81	5.49
Avg. Best	0.0001	0.0002		−1.96	1.16	5.89	−4109.43	0.182
Significance			> *chc*					
				CHC Algorithm				
% Solved	100%	100%	100%	100%	100%	0%	100%	43%
Avg. Time	20.21	7.18	0.51	27.02	1.91	1525.86	16.50	704.31
σ	20.48	9.62	0.07	11.24	1.96	16.10	4.66	210.02
Avg. Best						0.175		0.020
Significance	> *rmhc*	> *rmhc*		> *rmhc*	> *rmhc*	> *rmhc*	> *both*	> *rmhc*
				Delta Coding Algorithm				
% Solved	100%	100%	100%	100%	100%	100%	100%	100%
Avg. Time	0.35	1.03	0.41	12.86	0.87	754.63	34.10	60.16
σ	0.14	0.65	0.29	6.01	0.56	52.08	8.67	19.19
Significance	> *both*	> *both*		> *both*	> *both*	> *both*	> *rmhc*	> *both*

Notes Overhead

- Overhead in Delta Coding is inefficient
  - RMHC takes only slightly more recombinations then Delta Coding for F1 (Grey Coding)
    - Delta takes twice as much time
- If the problem is more difficult, then Delta is much faster (see table 10)
- Delta coding has more overhead then CHC in cases
  - Dramatic difference is related to the number of reinitializations

Conclusions

- CHC is very competitive with other algorithms and performs better in then all-in certain conditions. Shows the algorithm is very robust.
- Performance of CHC and Delta shows that restarts during genetic search are an effective method for maintaining population diversity.
  - Earlier study suggest high initial population and mutation rate
  - ESGA and GENITOR is only effective when RMHC performs well
  - Thus, might indicate populations converge quickly and depends on mutation
- Delta Coding performs consistently well with and without Grey Coding

Conclusions

- The suite of test problems should be reviewed
  - RMHC perform better then GA in several functions
    - However, RMHC do not always perform well in real-world problems and GA are needed when simpler methods fail
    - Argue that a good test suite should be resistant to simple stochastic hill-climbing algorithms
- GA are most useful when other simpler methods fail
  - So if stochastic hill climbing methods can solve the problem relatively easy, then the power of GA is not tested enough
- There is no theoretically reason to expect that Grey coding an arbritrarily function will locate the optimal solution easier for GA