Chapter 5: ENCODING

5.1 Prologue
In real world applications, the search space is defined by a set of objects, each of which has different parameters. The objective of optimisation problem working on these parameters is to optimise them. Genetic algorithms are also categorised as optimisation algorithms. Every search and optimisation algorithm needs a representation which represents a solution to a specific problem. Likewise, Genetic algorithms work on coding space and solution space alternatively. Genetic algorithms are biologically inspired, so their having two spaces is supported by natural evidence. In nature, coding space refers to the genotypic space and solution space refers to phenotypic space. This had been clearly investigated by Mendel in 1866. Mendel recognised that nature stores the complete genotype in number of strings called chromosomes. The genetic information in chromosomes is formed by double string of four nucleotides called DNA. The genotype represents all the information stored in the chromosomes and allows us to describe an individual on the level of genes [Mendel 1866]. The phenotype describes the outward appearance of an individual. A transformation exists between genotype and phenotype, also called mapping, which uses the genotypic information to construct the phenotype. A chromosome refers to a string of certain length where all the genetic information of an individual is stored. Each chromosome consists of many alleles. Alleles are the smallest information units in a chromosome [Holland 1975]. If a phenotypic property of an individual, like its eye colour is determined by one or more alleles, then these alleles together are denoted as gene as shown in Figure 1. A gene is a region on a chromosome which is responsible for a specific phenotypic property [Rothlauf 2006].

![Gene-Allele Representation in Chromosome](image)

Representation can also be termed as encoding. A representation assigns genotypes to corresponding phenotypes. In genetic algorithms, genetic operators work on the level of
genotype and evaluation of individuals is performed on the level of phenotype [Gen 1996]. So, certain mapping or coding function between the phenotypic space and the genotypic space is needed. This can be done by designing the representation as close as possible to characteristics of phenotypic space. The mapping of the object variables to a string code is achieved through an encoding function and the mapping of a string code to its corresponding object variable is achieved through a decoding function.

Genetic algorithms work on coding space and solution space alternatively. Genetic operations work on coding space i.e. chromosomes, while evaluation and selection work on solution space [Gen 1996]. The mapping of the object variables to a string code is achieved through an encoding function and the mapping of a string code to its corresponding object variable is achieved through a decoding function.

![Figure 5.2: Coding Space and Solution Space Representation](image)

Genetic algorithms encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality [Deb 1997]. These strings are referred to as chromosomes.

Encoding should be able to represent all possible phenotypes. It should encode no infeasible solutions. It should be unbiased. Decoding from phenotype to genotype should be easy. Problem should be represented at the correct level of abstraction.

### 5.2 Types of Encoding

**5.2.1 Binary Encoding**

Binary Encoding is the most common representation of chromosome in genetic algorithms. A binary string is defined using a binary alphabet \{0, 1\}. Each object variable is encoded in a
binary string of a particular length \( l_i \), defined by the user [Deb 1997]. Thereafter, a complete \( l \)-bit string is formed by concatenating all sub-strings together. Thus, the complete GA string has a length \( l \):

\[
l = \sum_{i=1}^{n} l_i \quad \text{(Eq. vi)}
\]

where \( n \) is the number of object variables. A binary string \( l_i \) has total of 2 \( l_i \) search points. The string length used to encode a particular variable depends on the desired precision in that variable. Sample representation of chromosome is

\[
101100101100101011100101
\]

Binary encoding allows for a higher degree of parallelism. It contains more schemas than decimal coding. In spite of these advantages, they are unnatural and unwieldy for many problems. They are prone to arbitrary orderings [Mitchell 1996]. Binary encoding does not provide the variety of options required to tackle the spectrum of problems faced in science, business and engineering. It contains more schemas than decimal coding. In binary encoding, different bits have different significance and hamming distance between two consecutive integers is often not equal to one [Eiben et al. 2003].

### 5.2.2 Gray Encoding

Ordinary binary number representation of the variable values may slow convergence of a GA. Increasing the number of bits in the variable representation magnifies the problem [Haupt et al. 1998]. Gray Code can avoid this problem by redefining the binary numbers so that consecutive numbers have a Hamming distance of one [Taub et al. 1986]. Gray codes speed convergence time by keeping the algorithm’s attention on converging toward a solution [Caruana et al. 1988]. Gray coding uses QUAD search for finding the solutions. As in binary strings, even in gray coded strings a bit change in any arbitrary location may cause a large change in the decoded integer value. The decoding of the gray coded strings to the corresponding decision variable introduces an artificial non-linearity in the relationship between the string and the decoded value.
5.2.3 Floating Point Encoding

Floating-point encoding scheme was developed by Deb for continuous variables [Deb 1997]. In this scheme, separate genes designated by M and E, respectively, represent both mantissa and exponent of a floating-point parameter. For a multi-parameter optimization problem, a typical gene has three elements, as opposed to two in earlier encoding schemes. The three elements are the parameter identification number, mantissa or exponent declaration, and its value. Various experiments have shown that the floating point encoding is simple to implement as it does not require any conversion mechanism that converts bit string to real value. Floating point encoding uses real coded genes. It is capable of incorporating various constraints in its implementation.

Various experiments have shown that the genetic algorithm with floating point encoding is simpler for implementation and faster than genetic algorithm with binary representation. This is because in case of binary implementation, the algorithm must have conversion mechanism that converts bit string to real value. Such mechanism is not required in case of floating point encoding. Floating point encoding uses real coded genes. It has been found that algorithm with floating point encoding gives better results than algorithm with binary encoding as it can easily incorporate various constraints.

5.2.4 Permutation Encoding

Permutation encoding is used in ordering problems, such as traveling salesman problem or task ordering problem. Given n unique objects, n! Permutations of the object exist. In this scheme, every chromosome is a string of numbers that represent a position (number) in a sequence. In certain cases random numbers between 0 and 1 are also used to encode the problem. These values are used as sort keys to decode the solution. Sample representation for two chromosomes is:

| Chromosome A | 1 5 3 2 6 4 7 9 8 |
| Chromosome B | 8 5 6 7 2 3 1 4 9 |

Permutation problems cannot be processed using the same general recombination and mutation operators that are applied to parameter optimization problems. Permutations are also
important for scheduling applications, variants of which are also often NP complete. This encoding is also called path representation or order representation [Starkweather et al. 1991]. Permutation problems cannot be processed using the same general recombination and mutation operators that are applied to parameter optimization problems. Permutations are also important for scheduling applications, variants of which are also often NP complete. This encoding is also called path representation or order representation.

5.2.5 Value Encoding

Value coding also called Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult. In the value encoding, every chromosome is a sequence of some values that can be anything connected to the problem, such as real numbers, chars or any objects. It is often necessary to develop some specific crossover and mutation techniques for these chromosomes [Meng 1996]. Sample representation is:

Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B ABDJEIFJDHDIERJFDLDFLFEFT

5.2.6 Tree Encoding

Tree encoding is used mainly for genetic programming. In the tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language. The representation space is defined by defining the set of functions and terminals to label the nodes in the trees. Trees provide rich representation that is sufficient to represent computer programs, analytical functions, and variable length structure, even computer hardware. Parse tree is a popular representation for evolving executable structures [Back et al. 1997]. Parse tree incorporates natural recursive definition, which allows for dynamically sized structures. Most of the parse tree representations have restriction on size of evolving programs. In a parse tree representation, the contents of the parse tree determine the power and suitability of the representation. Due to acyclic nature of parse trees, iterative computations are not naturally represented. It is very difficult to identify the stopping criteria. So the evolved function is evaluated within an implied loop that re-executes the evolved function until some predetermined stopping criteria is satisfied.
Tree encoding allows search space to be open ended. But due to open endedness, tree may grow in an uncontrolled way. Large trees are difficult to understand and simplify. Large trees also prevent structure and hierarchical candidate solutions. Parse tree incorporates natural recursive definition, which allows for dynamically sized structures. Most of the parse tree representations have restriction on size of evolving programs. If there is no restriction, then it would lead to increase in size of evolving programs and further lead to swamping of available computational resources. Size restriction is implemented in two ways. Depth limitation restricts the size of evolving parse tree based on user-defined maximal depth parameter. Node limitation places limit on total number of nodes available for an individual parse tree. Node limitation is preferred over size restriction because it encodes fewer restrictions on structural organization of evolving programs [Angeline 1996].

### 5.2.7 Messy Encoding

If in a problem, a particular bit combination for some widely separated genes constitute a building block, it will be difficult to maintain the building block in the population under the action of the recombination operator. This problem is largely known as the linkage problem in GAs. In order to solve the linkage problem, Messy coding was suggested by Goldberg *et al.* [Goldberg *et al.* 1989a]. Both the gene position and the corresponding bit values are coded in a string. A typical four-bit string is coded as follows:

\((2 \ 1) \ (4 \ 0) \ (1 \ 1) \ (3 \ 1))\)

The first entry inside a parenthesis is the gene location and the second entry is the bit value for that position. Since the gene location is also coded, good and important gene combinations can
be expressed tightly. This will reduce the chance of disruption of important building blocks
due to the recombination operator. Thus, it will have a lesser chance of disruption due to the
action of the recombination operator. This encoding scheme has been used to solve deceptive
problems of various complexities [Goldberg et al. 1989a].

5.2.8 Non-Binary Encoding
Although the binary strings have been mostly used to encode object variables, higher-ary
alphabets have also been used in some studies. For a \( \chi \)-ary alphabet string of length \( l \), there are
a total of \( \chi^l \) strings possible. Although the search space is larger with a higher-ary alphabet
coding than with a binary coding of the same length, Goldberg has shown that the schema
processing is maximum with binary alphabets [Deb 1997].

Prior to Holland, who invented the most common binary coded strings; non-binary alphabets
were used to code strings. The string length or length of chromosome was finite and used
minimal number of non-binary alphabets. Bagley used non-binary alphabets to represent
chromosome in diploid form in Hexapawn game playing. Rosenberg also used non-binary
alphabets to represent chromosome in diploid form in implementing genetic algorithm for
biological cell simulation [Goldberg 1989].

5.2.9 Encoding of Graphs using Vectors
Graph coloring problem has wide applications in contemporary problems with exponential
complexity. The objective of this problem is to determine the minimum number of colors
needed to color a given graph. A chromosome for this problem is encoded using a simple
vector whose length is equal to the number of vertices in the graph. Every gene or its position
in the graph corresponds to a vertex color [Harmanani et al. 2006].

5.2.10 Edge and Vertex Encoding
Edge and Vertex encoding also termed as link and node biased encoding, developed by
Palmer, was used for chromosome representation of combinatorial optimisation problem like
minimum spanning tree. In this encoding, the chromosome holds a bias value for each node
and link. Each link bias and each node bias are an integer in the range from 0 to 255. This
encoding does not directly encode a tree, but just a modified cost matrix. This method requires long encoding thus leading to higher memory cost. This method does not depict any information regarding degree, connection related to tree. It requires conventional minimum spanning tree algorithm to generate tree and thus leading to higher computational cost [Gen 1996].

5.2.11 Sparse Matrix Representation

Sparse Matrix representation (SMR) is used to represent information for problems like Production scheduling which may be of any form – flow shop, job shop or open shop scheduling. SMR generates feasible schedules over all evolutionary generations and no fixing scheme is required to ensure the validity of solutions. It is an allelic representation as it offers the facility to provide more information for complex problems and can also make GA operators more independent of the type of parameters. SMR relaxes the fixed locus limitation of conventional genetic representations and makes a flexible interpretation possible. It is a powerful representation as it maintains the sequence order and constraints associated with jobs being scheduled resulting in valid schedules only being generated as candidate solutions. It is an efficient and flexible method of representing production scheduling problems [Liang et al. 1995].

![Figure 5.4: 3 X 3 Job Scheduling problem](image1)

![Figure 5.5: Sparse Matrix Representation of 3 X 3 Job Scheduling](image2)
5.2.12 **Grammar based Encoding**

Grammar based Encoding is used in problems dealing with optimisation of automata networks. The global dynamics of automata networks (such as neural networks) are a function of their topology and the choice of automata used. Evolutionary methods can be applied to the optimisation of the parameters, but their computational cost is prohibitive unless they operate on a compact representation. Earlier direct coding was used as it was simple. Grammar-based encoding schemes use productions for genes, the expression of which forms the phenotype [Vonk et al. 1995]. The indirectness of this encoding allows for a many-to-one mapping with a corresponding richness in possible genotypic representations. Graph grammars provide such a representation by allowing network regularities to be efficiently captured and reused.

5.3 **Schemata Theorem**

Schemata were first proposed by Holland in 1975 to model the ability of Genetic algorithms to process similarities between bit strings [Holland 1975]. A schema is a subset of the space of all possible individuals for which all the genes match the template for schema. A schema \( h = (h_1, h_2, \ldots, h_l) \) is defined as a ternary string of length \( l \), where \( h_i \in \{0, 1, *\} \). * denotes the “don’t care” symbol and tells us that the allele at this position is not fixed. The size or order \( o(h) \) of a schema \( h \) is defined as the number of fixed positions (0s or 1s) in the string or the number of non *" genes in the schema. A position in a schema is fixed if there is either a 0 or a 1 at this position. The defining length \( \delta(h) \) of a schema \( h \) is defined as the distance between the two outermost fixed bits i.e. first and last non *" gene. For cardinality \( k \), there are \((k+1)^l\) schema in string of length ‘l’. The fitness of a schema is defined as the average fitness of all instances of this schema and can be calculated as:

\[
f(h) = \frac{1}{||h||} \sum_{x \in h} f(x)
\]

(Eq. vii)

where \( ||h|| \) is the number of individuals \( x \in \Phi_g \) that are an instance of the schema \( h \). The instances of a schema \( h \) are all genotypes where \( xg \in h \) [Rothlauf 2006].
Based on the notion of schemata, Holland formulated the schema theorem which describes how the number of instances of a schema $h$ changes over the number of generations $t$:

$$m(h, t + 1) \geq m(h, t) \frac{f(h, t)}{\bar{f}(t)} \left(1 - p_c \frac{\delta(h)}{l} - p_m o(h)\right)$$

(Eq. viii)

where

- $m(h, t)$ is the number of instances of schema $h$ at generation $t$,
- $f(h, t)$ is the fitness of the schema $h$ at generation $t$,
- $\bar{f}(t)$ is the average fitness of the population at generation $t$,
- $\delta(h)$ is the defining length of schema $h$,
- $p_c$ is the probability of crossover,
- $p_m$ is the probability of mutation,
- $l$ is the string length,
- $o(h)$ is the order of schema $h$

Schema theorem was formalised by John Holland and popularised by David Goldberg [Goldberg 1989].

5.4 Building Block Hypothesis

Basic building block can be defined as a short schema with sampling fitness that is greater than the sampling fitness of schemata. In other words, Schemata with high fitness values and small defining are called building blocks. The building block hypothesis states that Genetic Algorithms have the ability to propagate building blocks. A synergistic intersection between small collections of basic building blocks is called a second level building block. A synergistic intersection between small collections of second level building blocks is called a third level building block and so on. By combining schemata of lower order which are highly fit, overall good solution can be constructed.

The building block hypothesis rests on assumptions that are at least as strong as following two assumptions:
- Abundance: A large number of basic building blocks exist.
- Hierarchical Synergism: Synergistic intersections between small collections of collocated building blocks at same level are common.

Building Block Hypothesis is currently the central dogma for the adaptive capacity of genetic algorithms. Genetic algorithms seek near-optimal performance through the juxta-position of short, low-order, high performance schemata called building blocks. Building block hypothesis is criteria of how genetic algorithm works. Goldberg has stated in work that a genetic algorithm achieves high performance through juxtaposition of short low order highly fit building blocks [Goldberg 1989]. Schema is highly fit if its average fitness is considerably higher than the average fitness of all strings in the search space. Low order, well defined, average fitness schemata will combine through crossover to form high order, above average fitness schemata [Sivanandam et al. 2007].

Building blocks are also associated with search space as they supply regularities that can be exploited by any search technique.

5.5 Levels of Building Blocks in Other Systems

Building blocks play a central role both in Darwin’s natural selection and in neo-Darwinism. At the level of genome, building blocks are specified by individual genes having cumulative additive effects. In genomics, linked group of genes serve as building blocks under crossover and interactions they encode can be highly non-linear. The fundamental component of genes - DNA is constructed from four nucleotide building blocks – A, G, C and T which is basically amino acid sequences formed from combination of twenty amino acids. Amino acids are structural components of protein molecules that turn genes ‘on’ or ‘off’ and ‘autocatalytic bio-circuits’. Organelles are constructed from bio-circuits and so on. Their fitness depends on the sequence of combinations of amino-acids. [Holland 2000]. Different recombinations of amino-acids yield change in fitness. Similarly, tree can be said to constitute of building blocks like leaves, branches, trunk etc. In Physics, successive levels of building blocks also exist – nucleons constructed from quarks, nuclei constructed from nucleons, atoms constructed from nuclei, molecules constructed from atoms and so on.
According to Holland, main characteristics of building blocks are that they must be easy to identify and they must be readily combined to form a wide variety of structures. Encoding is first and the most critical issue in GA. The main question that arises when solving any problem by GA is how to encode various parameters of the problem into a chromosome. These artificial chromosomes can be strings of 1s and 0s, parameter lists, permutation codes or even more complex codes. GAs are robust and work quite well on arbitrarily chosen encoding. They exploit similarities in different encoding as long as building blocks lead to near optima.

### 5.6 Principles for Building Blocks

Genetic algorithms follow two basic principles for choosing the encoding method namely:

- **The principle of meaningful building blocks**: The schemata should be short, of low order, and relatively unrelated to schemata over other fixed positions.

- **The principle of minimal alphabets**: The alphabet of the encoding should be as small as possible while still allowing a natural representation of solutions.

The first principle states that the user should select a coding such that the building blocks of the underlying problem are small and relatively unrelated to building blocks at other positions. The principle of meaningful building blocks is directly motivated by the schema theorem. If schemata are highly fit, short and of low order, then their numbers exponentially increase over the generations. If the high-quality schemata are long or of high order, they are disrupted by crossover and mutation and they can not be propagated properly.

The second principle states that the user should select the smallest alphabet that permits an expression of the problem so that the number of exploitable schemas is maximized [Goldberg 1989]. The principle of minimal alphabets tells us to increase the potential number of schemata by reducing the cardinality of the alphabet. When using minimal alphabets the number of possible schemata is maximal. This is the reason why Goldberg advises us to use bit string representations, because high quality schemata are more difficult to find when using alphabets of higher cardinality.
These two principles of representations are based on the assumption that genetic algorithms process schemata and building blocks.

5.7 Goldberg’s Design Decomposition

Using Holland’s notion of schema and building blocks, Goldberg proposed decomposing the problem of designing a competent selecto-recombinative GA into seven subproblems [Goldberg 2002]:

1. Know what GAs process—building blocks: The primary idea of selectorecombinative GA theory is that genetic algorithms work through a mechanism of decomposition and re-assembly. The basic idea is that GAs (1) implicitly identifies building blocks or subassemblies of good solutions, and (2) recombines different subassemblies to form very high performance solutions.

2. Know the BB challengers—building-block-wise difficult problems: Problems that are hard have BBs that are hard to acquire. This may be because the BBs are deep or complex, hard to find, or because different BBs are hard to separate, or because low-order BBs may be misleading.

3. Ensure an adequate supply of raw BBs: BBs are treated as a kind of material quantity that must be transported through space and time. One role of the population is to ensure adequate supply of the raw building blocks in a population.

4. Ensure increased market share for superior BBs: Another key idea is that BBs or notions exist in a kind of competitive market economy of ideas, and steps must be taken to ensure that the best ones (1) grow and take over a dominant market share of the population, and (2) the growth rate can neither be too fast, nor too slow.

5. Know BB takeover and convergence times: Thereafter, it considers how long convergence takes on average (convergence time). Randomly generated populations of increasing size will, with higher probability, contain larger numbers of more complex BBs.

6. Make decisions well among competing BBs: It ensures that the pool of choices is sufficiently rich to permit good solutions (the decision question).

7. Mix BBs well: Identification and exchange of BBs is the critical path to innovative success. Finally, it makes sure that different building blocks come together on the same string through effective exchange (BB mixing). First-generation GAs, usually fail in their ability to promote this exchange reliably. The primary design challenge to achieving competence is the
need to identify and promote effective BB exchange. Efforts in principled design of effective
BB identification and exchange mechanisms have led to the development of competent
genetic algorithms.

5.8 Crossover Operators for Different Representations
Binary encoding is the simplest method of representing chromosome. In crossover operation,
it takes two chromosomes as parent and generates two offspring chromosomes. Three
different forms of crossover suitable for binary encoding are one point crossover, N point
crossover and Uniform crossover. Chromosomes having Real value or Floating point
representation undergo Arithmetic Crossover. This crossover creates a new allele at each
gene position in the offsprings. The value of new allele lies between the values of the parent
alleles. The value of new alleles for offsprings is computed using following equation:

\[
\text{Offspring}_1 = w\text{parent}_1 + (1-w)\text{Parent}_2 \\
\text{Offspring}_2 = (1-w)\text{parent}_1 + w\text{Parent}_2 
\]

where \( w \) is constant weight factor that is used to compute new values.

Encoding having integer values in their representation also perform the same set of crossover
operations as performed by binary encoding such as One point crossover, N point crossover,
Uniform crossover. Special crossover operators are need to perform crossover operation on
orderd chromosomes [Eiben et al. 2003] having permutation encoding such as Edge
Crossover, Partially mapped (PMX) crossover, Order crossover and Cycle crossover.

5.9 Mutation Operators for Different Representations
Flip bit is very simple mutation used for binary encoding. In this, bit 0 changes to 1 or bit 1
changes to 0. The number of bits that change depends on the mutation rate. For Real value or
Floating point representation, there exists uniform mutation, Gaussian mutation and
Boundary mutation. Special mutation – Creep occurs in case of integer representation. For
permutation representation, there are variety of mutation operators like Swap, Scramble and
Inversion [Eiben et al. 2003].
5.10 Inversion

It is the most important and widely used mutation operator that is mainly used in ordered chromosome. It selects two positions on the chromosome randomly and reverses the order of values between these two positions. Inversion operation can change the location of character in a string. Any kind of arrangement of characters in a string can be obtained by using related inversion operations. It randomly selects two locations and inverses the elements between the two locations to form the new offspring. The benefit of inversion is that it brings certain alleles together or closer. It also helps in establishing linkage between better alleles in the chromosome. In terms of schema, the main influence of inversion operation on schema H is to randomly change the length of schema H and the relationship among effective characters in schema H [Zhonzhi 2011]. Inversion operator is suitable to chromosomes represented by permutation encoding. But when inversion is applied to other representations, alleles must accompanied by markers and decoding of strings is needed at the end [Koza 1992]. In case of binary encoding, inversion is not applicable as it would change the basic principle of binary encoding.

5.11 Encoding Schemes - Categorisation

Depending on the structure of encoding, it can be classified into two categories, namely, one dimensional and two-dimensional. Binary, Value, Real value and Permutation encoding is one dimensional and Tree encoding is two dimensional encoding techniques [Rothlauf 2006]. Studying these encoding schemes, one can infer that characters represented by permutation encoding are position dependent. In Binary encoding, real value encoding, the characters are value oriented. The two factors identified by studying different encoding schemes are locus and value of character in the chromosome. So, factors like locus and value should be kept in mind while encoding a solution for a particular problem. Binary encoding is the simplest representation and supports various types of crossover operations. It does not support inversion operator as the chromosome is not ordered and implying inversion on binary representation would result in disruption of building blocks and change in fitness value. Integer and Floating point representations have limited application depending on the problem. Permutation encoding is used to represent certain order in chromosomes. It supports inversion operator but does not support simple crossover operations like one point crossover,
n point crossover. Special crossover operators like PMX, order or cycle crossover is used for this representation making its use quite complex.

5.12 Goldberg’s Categorization and Need of Proposed Encoding

Goldberg has stated in his work that fitness function for a specific encoding scheme is dependent on two factors – value and order [Goldberg 1989]. Three different categories of encoding can be grouped depending on fitness evaluation factors such as:

- Encoding schemes where fitness depends on value only : \( f(v) \). Eg: Value encoding
- Encoding schemes where fitness depends on value and order : \( f(v,o) \). Eg: Binary Encoding
- Encoding schemes where fitness depends on order only : \( f(o) \). Eg: Permutation encoding.

It can be stated that the existing encoding schemes fall under these three categories and they are dependent on value or order or both factors for evaluation of fitness function.

Seeing this, there arises a need to unearth a new encoding scheme that is independent of these two factors. In this chapter, a naive encoding scheme is put forward that evaluates fitness of a chromosome on individual contribution of gene and not depending upon its value or order.

5.13 Proposed BCD Value Encoding

Seeing the limitations of binary encoding and permutation encoding, a new encoding scheme that aims to overcome these limitations and converge the advantages of both the schemes into one scheme. This novel approach of encoding is based on the concept to retain good building blocks in the chromosome regardless of their locus. The novel encoding scheme also has its biological justification. As in case of human beings, the presence of genes on the chromosomes is independent of its location. Genes in human chromosomes can be shifted from locus to other. The gene or set of genes is considered to be good building block on the basis of its contribution to fitness value.
In binary encoding, the genes are positional in nature and contribute to fitness according to its position in the chromosome. For example if we consider chromosome 10110, then the fitness of chromosome is 22. But if any of the bit changes its position, the fitness value also changes accordingly. For example, fitness of 11010 is 26. This clearly shows that the genes in binary encoding have positional significance.

The novel encoding scheme proposes to represent each gene in the chromosome by its fitness contribution instead of 1 or 0 as in binary encoding. Fitness contribution of each gene is computed as the 8421 scheme used in binary number system. For example, 10110 in binary encoding could be represented as 16 0 4 2 0 as per the new encoding scheme. Due to this reason, the proposed encoding scheme has been coined the term BCD value encoding.

The use of fitness contribution itself as gene instead of allele value 0 or 1 aids in locating good building blocks in the chromosome easily. Genes with higher fitness contribution are more likely to be selected as good building blocks and carried forward to next generations by any of the genetic operation. The proposed BCD value encoding allows inversion of genes without affecting the fitness of chromosome which would help in grouping or bringing closer good building blocks and develop linkage between them. The scheme also retains the capability of binary encoding to undergo one-point crossover or N-point crossover. In case, there occurs some redundancy of genes in chromosome during crossover, then any one of the redundant gene could be chosen and the other is replaced by 0. The position of genes is independent of fitness of chromosome.

For example, 16 0 4 0 1, 4 16 0 0 1, 0 0 4 1 16 and 1 0 16 0 4

These are various representation of chromosomes having same fitness value i.e. 21 but the locus of genes is different in each case.

### 5.14 Benefits of Proposed encoding

Benefits of proposed BCD value encoding scheme are numerous. It allows inversion operation despite using binary alleles in formation of chromosomes initially. It can undergo all types of crossover that a binary one could do. It helps in generating more fit individuals as
it supports retaining of good building blocks with higher fitness contribution. This is done by grouping of good building blocks by reducing the hamming distance between them. This encoding scheme can be applied to all the applications where initially binary encoding is implemented and would prove as efficient method of encoding.

The proposed BCD value encoding scheme based on fitness contribution has proven to be better than permutation encoding in grouping low order and highly fit schemas and works well on any type of crossover operation. Moreover, Inversion is an added advantage in this scheme over binary encoding that helps in grouping highly fit schemas together.

5.15 Implementation

To test the viability and usage of the proposed BCD value encoding, it was tested on fitness function used in Standard Genetic Algorithm (SGA) i.e. \( f(x) = x^2 \). MATLAB code has been developed to test the performance of genetic algorithm using roulette wheel selection in four cases.

- Case 1: Binary encoding and one point crossover
- Case 2: Proposed BCD value encoding and one point crossover
- Case 3: Proposed BCD value encoding and PMX crossover
- Case 4: Proposed BCD value encoding and Inversion

Case 1 tests the performance of binary encoding with one point crossover just like SGA. In existing encoding schemes, it has been observed that they are not suitable (apt) for all types of crossover operators. The proposed BCD value encoding has been tested with one point crossover (Case 2) that can be used with binary encoding and has proved to be suitable. Further, the proposed BCD value encoding has been used with the PMX crossover (Case 3) which is specifically used with permutation encoding. The proposed BCD value encoding has been found apt for PMX crossover as well and the test runs confirmed the proposition. To add an extra advantage, proposed BCD value encoding was tested with inversion operator also (Case 4). Inversion operator was confined to specific encoding schemes earlier. Test runs have confirmed the suitability of the proposed BCD value encoding with inversion operator too.
Demonstration of proposed encoding is illustrated for four different cases in this section.

Case 1: Binary encoding with one point crossover

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fitness value = 15
Fitness value = 22
Fitness value = 14
Fitness value = 23

Child chromosomes are created after one point crossover.

Case 2: Proposed BCD value encoding with one point crossover

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>16</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>C1</td>
<td>16</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Fitness value = 22
Fitness value = 15
Fitness value = 23
Fitness value = 14

As per proposed encoding, the alleles contain the value corresponding to the fitness contribution. After one point crossover, the two child chromosomes are created as illustrated. This shows that proposed encoding supports one point crossover as in binary encoding and can be used with n-point crossover also.

Case 3: Proposed BCD value encoding with PMX crossover

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>16</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>C1</td>
<td>16</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Fitness value = 22
Fitness value = 15
Fitness value = 30
Fitness value = 7

Highlighted parts of the parent chromosomes represent the swab portion of chromosome which is copied as such in PMX crossover. After PMX crossover, the two child chromosomes...
are created as shown. This shows that proposed encoding supports PMX crossover as in permutation encoding because fitness value of chromosome as per proposed encoding is not dependent on position of allele.

Case 4: Proposed BCD value encoding with Inversion

P1 0 8 4 2 1  Fitness value = 15
P2 16 0 4 2 0  Fitness value = 22

After Inversion

P1 0 4 2 1 8  Fitness value = 15
P2 16 4 2 0 0  Fitness value = 22

After one point crossover

C1 0 4 2 0 0  Fitness value = 6
C2 16 4 2 1 8  Fitness value = 31

Inversion is beneficial genetic operator which allows to group together good building blocks in a chromosome. Inversion operation cannot be applied to binary encoding as it is value dependent and cannot be applied to permutation encoding as it is order dependent. Since, the proposed BCD value encoding is independent of value and order, inversion operator can be applied to the chromosome. This would lead to grouping of good alleles which would lead to better offsprings. Fitness value of chromosome as per proposed BCD value encoding is independent of position, so inversion does not affect the computation of fitness value.

Test runs for standard genetic algorithm have been implemented using MATLAB code. Comparison of four given cases is done for average fitness and maximum fitness and is shown in Fig.5.6 and Fig.5.7.
It has been affirmed by the test runs that proposed encoding is not confined to specific set of genetic operators. It has wide range of applicability and independence of genetic operator to use. The proposed encoding is independent of value and order, so it can be used with any type crossover or inversion operator. It can also be tested with different mutation operators. It helps in retaining good building blocks with higher fitness contribution as it supports
inversion operator which was lacking in permutation encoding. Proposed encoding is independent of order of alleles in the chromosome, so it can be used with PMX crossover and can also be tested with order crossover etc.

5.16 Summary

The chapter reviews various encoding schemes and categorises existing encoding schemes as a function of value or order or both. Different crossover and mutation operators as per respective encoding scheme have been discussed in the chapter. The chapter also discusses the Holland’s Schema Theorem and building block hypothesis. A new encoding scheme is proposed in the chapter which is independent of function of order or value. Genes are represented by their respective contribution of fitness and is independent of order of genes in the chromosome. The proposed encoding is suitable for both one point crossover and PMX crossover and supports inversion operation also. Significance of proposed encoding is demonstrated by testing its utility in genetic algorithm using MATLAB code and test runs confirm its worth and independent behaviour.