Genetic Algorithms – a Tutorial

A.Townsend 13/07/03

Contents

1 Introduction
  1.1 Evolutionary Computing
    1.1.1 Evolutionary algorithms
      1.1.1.1 Genetic Algorithms
2 Biological Terminology
  2.1 General Concepts
  2.2 Molecular Biology
    2.2.1 DNA
    2.2.2 RNA
    2.2.3 Genes
    2.2.4 Chromosomes
3 Genetic algorithm basics
  3.1 Genetic algorithms overview
  3.2 Structure of a single population GA
    3.2.1 GA operators
  3.3 Multipopulation Evolutionary Algorithms
  3.4 Properties of GAs
    3.4.1 Most important parameters
    3.4.2 Considerations given to the use of GAs
    3.4.3 Some properties of GAs
    3.4.4 Differences between search methods
  3.5 GA Vocabulary
4 GA components
  4.1 A GA example
  4.2 Encoding
    4.2.1 Unsigned Fixed Point representation
    4.2.2 Genotypes and Phenotypes revisited
    4.2.3 Determining the chromosome length
    4.2.4 Determining the fixed point value from the chromosome
  4.3 Initialization
  4.4 Genetic Operators
    4.4.1 Selection
      4.4.1.1 Fitness Assessment
        4.4.1.1.1 Deterministic problems
        4.4.1.1.2 Non-Deterministic problems
      4.4.1.2 Selection and Breeding
        4.4.1.2.1 Convergence
        4.4.1.2.2 High-level approaches used in EC
        4.4.1.2.3 Parent Selection – Mating Pool
          4.4.1.2.3.1 Explicit fitness remapping
          4.4.1.2.3.2 Implicit fitness remapping
        4.4.1.2.4 Fitness-Proportional Selection
        4.4.1.2.5 Ranked Selection
        4.4.1.2.6 Roulette-wheel Selection
        4.4.1.2.7 Stochastic Universal Sampling

4.4.1.2.8 Truncation Selection
4.4.1.2.9 Tournament Selection
4.4.2 Recombination or Crossover
4.4.2.1 Binary valued recombination
4.4.2.1.1 One-point Crossover
4.4.2.1.2 Two-point Crossover
4.4.2.1.3 Uniform Crossover
4.4.2.1.4 Shuffle Crossover
4.4.2.1.5 Partially matched crossover (PMX)
4.4.2.1.6 Order Crossover (OX)
4.4.2.1.7 Cycle crossover (CX)
4.4.2.1.8 Other Crossover Techniques
4.4.2.2 Real valued recombination
4.4.2.2.1 Discrete recombination
4.4.2.2.2 Intermediate recombination
4.4.2.2.3 Line recombination
4.4.2.3 Summary
4.4.3 Mutation
4.4.3.1 Real-valued mutation
4.4.4 Linkage and Inversion
4.4.4.1 Inversion
4.4.5 Elitism
4.5 Some basic theorems and terms
4.5.1 Schemata and the Schema theorem
4.5.2 Implicit Parallelism Theorem
4.5.3 Building Block Hypothesis
4.5.4 Epistasis
4.5.5 Deception
4.6 Implementation Guidelines
4.7 References
Appendix 1

1 Introduction

Knowledge-based information systems or Evolutionary computing algorithms are designed to mimic the performance of biological systems. Evolutionary computing algorithms are used for search and optimization applications and also include fuzzy logic, which provides an approximate reasoning basis for representing uncertain and imprecise knowledge. As the name implies, artificial neural networks mimic the brain or biological information processing mechanisms. These they do in a very limited sense. The no free lunch theorem states that no search algorithm is better on all problems. Know from this no free lunch theorem that all search methods show on average the same performance over all possible problem instances. The present trend is to combine these fields into a hybrid in order that the vagaries of one may be offset by the merits of another. Neural networks, fuzzy logic and evolutionary computing have shown capability on many problems, but have not yet been able to solve the really complex problems that their biological counterparts can. Some of these hybrid techniques are,

- Evolutionary algorithm parameters (population size, selection, etc.) controlled by fuzzy systems,
- Neural network parameters (learning rate) controlled by fuzzy systems,
- Fuzzy logic controllers generated and tuned by evolutionary algorithms,
- Fuzzy logic controllers tuned by neural networks,
- Evolutionary computing in automatically training and generating neural network architectures

Figure 1, shows where the field of genetic algorithms is placed in the hierarchy of knowledge based information systems or evolutionary computing. This tutorial considers only genetic algorithms.

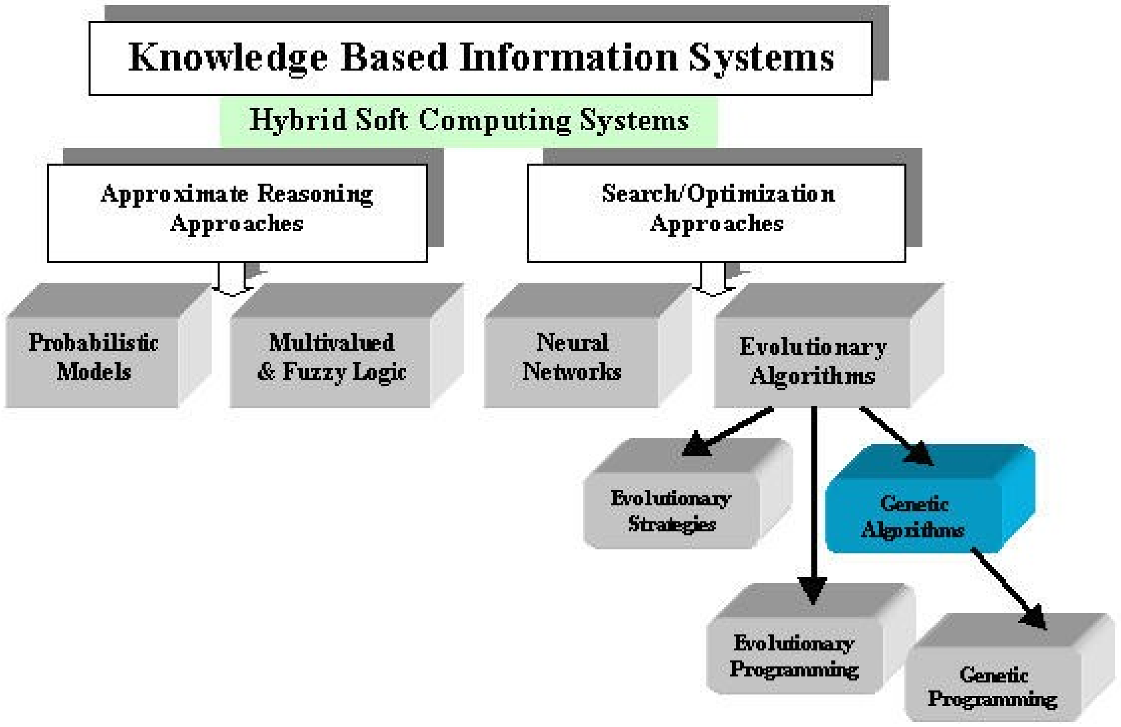

Knowledge Based Information Systems
Hybrid Soft Computing Systems
Approximate Reasoning Approaches
Search/Optimization Approaches
Probabilistic Models
Multivalued & Fuzzy Logic
Neural Networks
Evolutionary Algorithms
Evolutionary Strategies
Genetic Algorithms
Evolutionary Programming
Genetic Programming

Figure 1 The placement of Genetic Algorithms in the hierarchy of Knowledge Based Information Systems (Evolutionary computing).

1.1 Evolutionary Computing

Soft computing (SC) is the symbiotic use of many emerging problem-solving disciplines. *In contrast to the traditional computing that is classed as "hard computing", soft computing exploits the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness, low solution-cost and better rapport with reality* (Prof. Zadeh). Evolutionary computation is the name given to a collection of algorithms based on the evolution of a population toward a solution of a certain problem. It is one of the four main components of soft computing, which are,

approximate reasoning

- Probabilistic Reasoning,
- Fuzzy Logic,

Search/Optimization,

- Neural Networks,
- Evolutionary Algorithms

1.1.1 Evolutionary algorithms

Evolutionary algorithms can be used successfully in many applications requiring the optimization of a certain multi-dimensional function. The population of possible solutions evolves from one generation to the next, ultimately arriving at a satisfactory solution to the problem. These algorithms differ in the way a new population is generated from the present one, and in the way the members are represented within the algorithm. They are part of the derivative-free optimization and search methods that comprise,

- Simulated annealing (SA) which is a stochastic hill-climbing algorithm based on the analogy with the physical process of annealing. Hill climbing, in essence, finds an optimum by following the local gradient of the function (thus, they are also known as gradient methods).
- Random Search Algorithms - Random searches simply perform random walks of the problem space, recording the best optimum values found. They do not use any knowledge gained from previous results and are inefficient.
- Randomized Search Techniques - These algorithms use random choice to travel through the search space using the knowledge gained from previous results in the search.
- Downhill simplex search
- Tabu search which is usually applied to combinatorial optimization problems

Four types of evolutionary algorithm techniques are presently being used. These are,

Genetic Algorithms (GAs),

Evolutionary Strategies (ES)[18],

- Originally proposed for the optimization of continuous functions
- Comprises recombination and mutation operators
- Focus is on the behavior of individuals

(ES) searches over fixed parameter spaces as does GAs. ES has traditionally applied Gaussian mutation to fixed-length vectors of real-valued numbers, using the (μ,λ) or $(\mu + \lambda)$ selection schemes (see section 4.4.1.2.6). ES often applies the 1/5 *rule* to determine the variance of the Gaussian distribution used in mutation: if more than one fifth of new children are fitter than their parents, then the variance is increased to prevent the search from exploiting local improvements too much. Conversely, if less than one fifth of new children are more fit than their parents, then the variance is decreased to keep the search from exploring too widely. ES also can use *self-adaptive mutation*, adding to each individual one or two additional parameters that indicate how mutation should be applied to the individual. In this way, individuals can evolve not only their candidate solutions but also the way in which the solutions are to be modified. ES traditionally uses only mutation, though ES researchers have recently adopted forms of crossover.

Genetic Programming (GP) [1]
- A special case of Genetic Algorithms
- Chromosomes have a hierarchical rather than a linear structure that is not limited in size.
- Their sizes are not predefined
- Individuals or chromosomes are tree-structured programs
- Genetic operators work on the branches of these trees, that is, modified operators are applied to sub-trees or single nodes
- Due to the tree structures, the computer programs are usually written in LISP

Evolutionary Programming (EP).
- Originally proposed for sequence prediction and optimal gaming strategies
- The genetic operators work directly on the actual structure
- The structures used are representations that are problem dependent and more natural for the task than the general representations used in GAs.
- actually manipulates entire computer programs, so the technique can potentially produce effective solutions to very large-scale problems
- To reach full potential, it will likely require improvements in computer hardware
- Currently focussed on continuous parameter optimization and training of NNs
- Could be considered as a special case of ES without recombination
- Focus is on the behavior of species

Fogel, Owens, and Walsh in1966 [19] argued that mutation should be the sole breeding mechanism. EP has no traditional genome representation—it proposes evolving whatever genome structure is most suitable for the task, as long as a mutation operator can be devised to modify it. EP is the oldest evolutionary computation field, and was originally devised to evolve individuals in the form of finite-state automata. These early experiments mutated individuals by directly adding and deleting vertices and edges to individuals' FSA graphs.

Figure 1, shows the placement of these computing techniques in the Evolutionary Algorithm hierarchy. All four of these algorithms are modeled in some way after the evolutionary processes occurring in nature.
Evolutionary algorithms exhibit an adaptive behavior that allows them to handle non-linear, high dimensional problems without requiring differentiability or explicit knowledge of the problem structure. They also are very robust to time-varying behavior, even though they may exhibit low speed of convergence.

1.1.1.1 Genetic Algorithms

Genetic algorithms are search methods that employ processes found in natural biological evolution. These algorithms search or operate on a given population of potential solutions to find those that approach some specification or criteria. To do this, the algorithm applies the principle of survival of the fittest to find better and better approximations. At each generation, a new set of approximations is created by the process of selecting individual potential solutions (individuals) according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

The GA will generally include the three fundamental genetic operations of selection, crossover and mutation. These operations are used to modify the chosen solutions and select the most appropriate offspring to pass on to succeeding generations. GAs consider many points in the search space simultaneously and have been found to provide a rapid convergence to a near optimum solution in many types of problems; in other words, they usually exhibit a reduced chance of converging to local minima. GAs suffer from the problem of excessive complexity if used on problems that are too large.

Figure 2 shows the structure of a simple genetic algorithm. Genetic algorithms work on populations of individuals rather than single solutions, allowing for parallel processing to be performed when finding solutions to the more large and complex problems. They are an iterative procedure that consists of a constant-sized population of individuals, each one represented by a finite linear string of symbols, known as the chromosome, encoding a possible solution in a given problem space. This space, referred to as the search space or state space, comprises all possible solutions to the optimization problem at hand. Standard genetic algorithms are implemented where the initial population of individuals is generated at random. At every evolutionary step, also known as *generation*, the individuals in the current population are decoded and evaluated according to a fitness function set for a given problem. The expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Crossover is performed between two selected individuals by exchanging part of their genomes to form new individuals. The mutation operator is introduced to prevent premature convergence.

Every member of a population has a certain fitness value associated with it, which represents the degree of correctness of that particular solution or the quality of solution it represents. The initial population of strings is randomly chosen. The GA using genetic operators, to finally arrive at a quality solution to the given problem manipulates the strings. GAs converge rapidly to quality solutions. Although they do not guarantee convergence to the single best solution to the problem, the processing leverage associated with GAs make them efficient search techniques. The main advantage of a GA is that it is able to manipulate numerous strings simultaneously by parallel processing, where each string represents a different solution to a given problem. Thus, the possibility of the GA getting caught in local minima is greatly reduced because the whole space of possible solutions can be simultaneously searched.

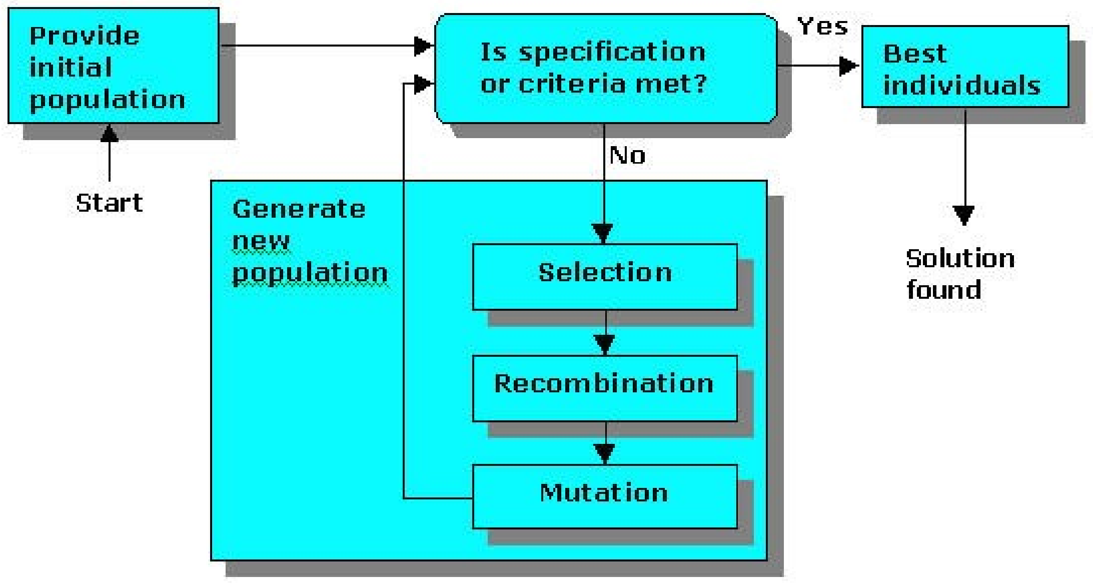


Figure 2 Structure of a single population evolutionary algorithm

In this tutorial, section 2 provides a short overview of the biological terms used throughout the tutorial. Section 3 provides an overview on the structure and basic algorithms of genetic algorithms. Section 4 describes the GA operators in more detail. In Section 5 the different recombination algorithms are presented. Section 6 explains mutation and Section 7 reinsertion. Section 8 covers parallel implementations of evolutionary algorithms especially the migration model in detail.

2 BIOLOGICAL TERMINOLOGY

2.1 General Concepts

Evolution is a cumulative process. *Inheritance* is the determinant of almost all of the structure and function of organisms since life began. The amount of variation from one generation to the next is quite small and some molecules, such as those that carry energy or genetic information, have seen very little change since the original common ancestor of several billion of years ago.
Inheritance alone does not give rise to evolution because pure inheritance would lead to populations of entirely identical organisms, all exactly like the first one.
In order to evolve, there must be something that causes a *variation* in the structure of the material that an organism inherits material from its parent or parents. In biology, there are several sources of variation. To name a few, mutation, or random changes in inherited material, sexual recombination and various other kinds of genetic rearrangements, even viruses can get into the act, leaving a permanent trace in the genes of their hosts. All of these sources of variation modify the message contained in the material that is passed from parent to offspring. It is an evolutionary truism that almost all variations are neutral or deleterious. Small changes in a complex system often lead to far-reaching and destructive consequences (the butterfly effect in chaos theory). However, given enough time, the search of that space that contains the organisms with their varied inherited material, has produced many viable organisms.
Selection is the determining process by which variants are able to persist and therefore also which parts of the space of possible variations will be explored. Natural selection is based on the reproductive fitness of each individual. Reproductive fitness is a measure of how many surviving offspring an organism can produce; the better adapted an organism is to its environment, the more successful offspring it will create.
Because of competition for limited resources, only organisms with high fitness survive. Those organisms less well adapted to their environment than competing organisms will simply die out.
Evolution can be likened to a search through a very large space of possible organism characteristics. That space can be defined quite precisely. All of an organism's inherited characteristics are contained in a single messenger molecule: deoxyribonucleic acid, or DNA. The characteristics are represented in a simple, linear, four-element code. The translation of this code into all the inherited characteristics of an organism (e.g. its body plan, or the wiring of its nervous system) is complex. The particular genetic encoding for an organism is called its *genotype*. The resulting collective physical characteristics of an organism is called its *phenotype*. In the search space metaphor, every point in the space is a genotype. Evolutionary variation (such as mutation, sexual recombination and genetic rearrangements) identifies the legal moves in this space. Selection is an evaluation function that determines how many other points a point can generate, and how long each point persists. The difference between genotype and phenotype is important because allowable (i.e. small) steps in genotype space can have large consequences in phenotype space. It is also worth noting that ***search happens in genotype space***, but ***selection occurs on phenotypes***. Although it is hard to characterize the size of phenotype space, an organism with a large amount of genetic material (like, e.g., that of the flower Lily) has about 10^{11} elements taken from a four letter alphabet, meaning that there are roughly $10^{70,000,000,000}$ possible genotypes of that size or less. A vast space indeed! Moves (reproductive events) occur asynchronously, both with each other and with the selection process. There are many non-deterministic elements; for example, in which of many possible moves is taken, or in the application of the selection

function. Imagine this search process running for billions of iterations, examining trillions of points in this space in parallel at each iteration. Perhaps it is not such a surprise that evolution is responsible for the wondrous abilities of living things, and for their tremendous diversity.
All of the genetic material in an organism is called its *genome*. Genetic material is discrete and hence has a particular size, although the size of the genome is not directly related to the complexity of the organism.

2.2 Molecular Biology

At this point it is useful to introduce and explain some of the biological terms used throughout this tutorial [2].
All living organisms consist of cells. Figure 3 shows a diagram of a cell.

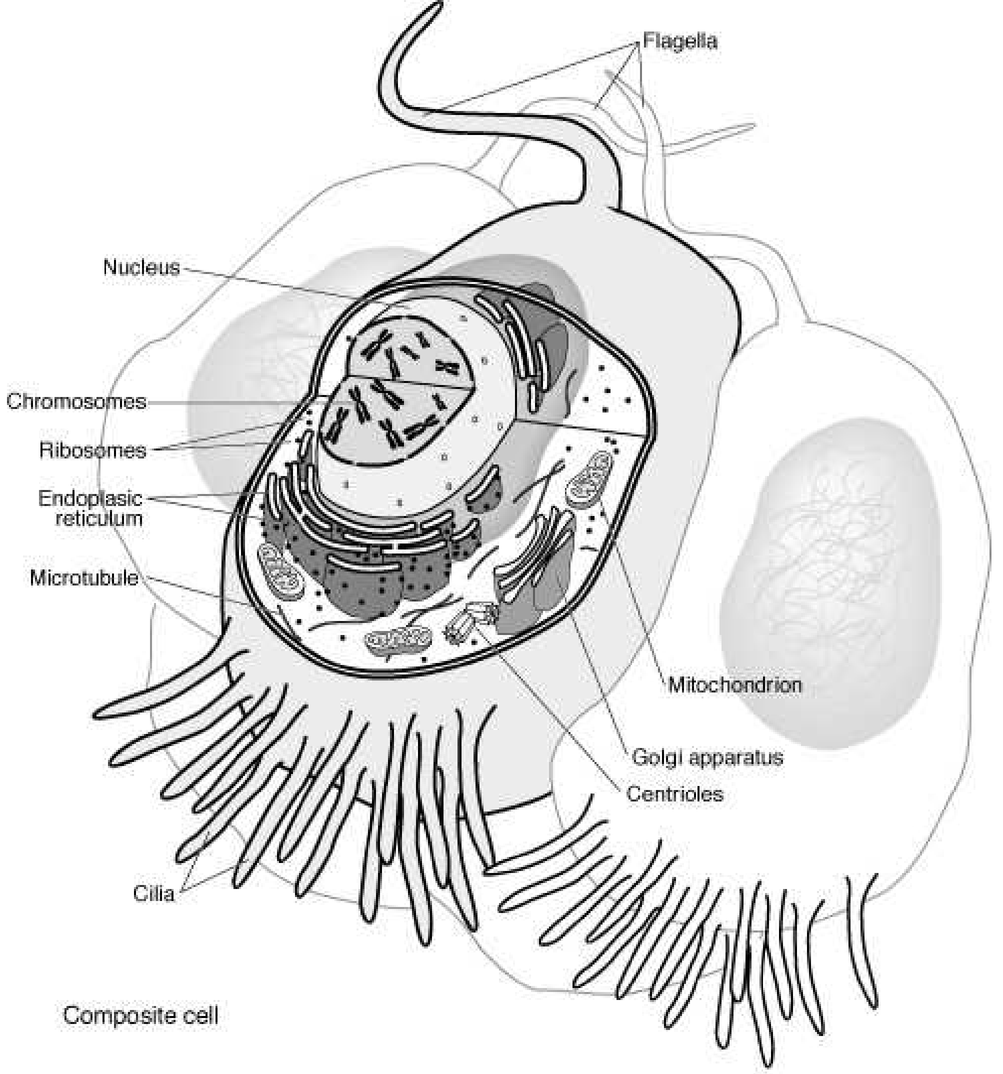


Figure 3 A Human cell

The function of a cell is to reproduce its DNA (*deoxyribonucleic acid*)[3] for growth and reproduction, or manufacture protein for cellular functioning. Once it has matured, its main purpose is to produce protein. It is the production of protein that gives the cell its unique characteristics. Such as a brain cell differs from a skin cell because of the proteins each one produces. All cells of an organism contain exactly the same DNA in exactly the same

arrangement. All cells in the human body are the same, it is only particular genes that are switched on in some cells and not in others that direct the manufacture of specific proteins. So, in a liver cell, the regions of DNA that make liver proteins, and produce liver enzymes, specialized tissues, etc. are in the "on" mode and all other regions that are outside of the liver are switched "off". In a tongue cell, the only regions switched on are those that make proteins needed for the tongue.
The complete set of instructions for making an organism is called its *genome*. The genome is defined as the entire DNA contained in an organism or a cell, which includes both the chromosomes within the nucleus and the DNA in mitochondria (where the cell energy comes from). This genome contains the master blueprint for all cellular structures and activities for the lifetime for the cell or organism.
Found in every nucleus of a person's many trillions of cells, the human genome consists of tightly coiled threads of ***deoxyribonucleic acid*** (DNA) as shown in Figure 4.

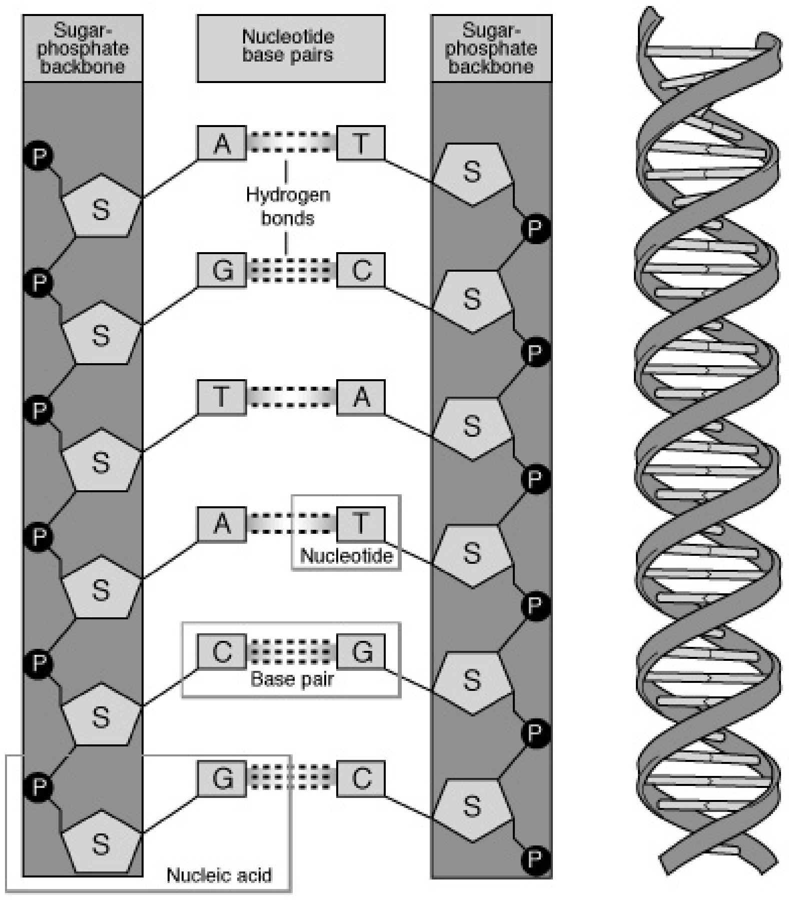
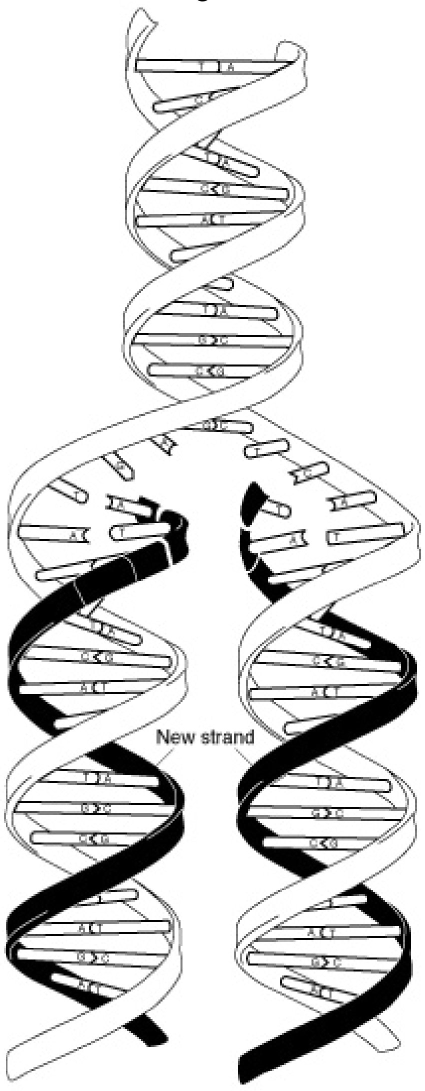


Figure 4 The structure of deoxyribonucleic acid (DNA)
The four nitrogenous bases of DNA are arranged along the sugar phosphate backbone in a particular order (the DNA sequence), encoding all genetic instructions for an organism. Adenine (A) pairs with thymine (T), while cytosine (C) pairs with guanine (G). The two DNA strands are held together by weak bonds between the bases. A ***gene*** *is a segment of a DNA molecule (ranging from fewer than 1 thousand bases to several million), located in a particular position on a specific chromosome whose base sequence contains the information necessary for protein synthesis.*

2.2.1 DNA

In all higher organisms, including humans, a DNA molecule comprises two strands that wrap around each other to resemble a double helix. The sides of this double helix are made of sugar and phosphate molecules and are connected by "rungs" of nitrogen-containing chemicals called ***bases***. Each strand is a linear arrangement of repeating similar units called ***nucleotides (nucleic acid)***, that are each composed of one sugar, one phosphate and a nitrogenous base as shown in Figure 4. These are the building blocks of DNA. Four different bases are present in DNA. These are adenine (A), thymine (T), cytosine (C) and guanine (G). From these four neucleotides, any of twenty-four amino acids can be built, each one coded by a three-neucleotide group, also known as a ***codon***. Thus an amino acid can be equal to ATC, TCA, CGT, … etc. From the side by side arrangement along the sugar-phosphate backbone of a few hundred to a few thousand of twenty of these amino acids, the cell is capable of stringing together long, folded proteins that accomplish all the functions of the organism. Thus, nucleotides produce amino acids that in turn



produce proteins.

Figure 5 DNA replication

The particular order of the bases arranged along the sugar-phosphate backbone is called the DNA sequence. This sequence specifies the exact genetic instructions required to create a particular organism with its own unique traits. This is why we are human beings and not birds.
The two DNA strands are held together by weak bonds between the bases on each strand, forming base pairs (bp). Genome size is usually stated as the total number of base pairs. The human genome contains roughly three billion base pairs.

Each time a cell divides into two offspring cells, its full genome is duplicated. For humans and other complex organisms, this duplication occurs in the nucleus. During cell division the DNA molecule unwinds and the weak bonds between the base pairs break, allowing the strands to separate. Each strand directs the synthesis of a complementary new strand, with free nucleotides floating around in the cell cytoplasm matching up with their complementary bases on each of the separated strands. This is shown in Figure 5. The procedure of copying one string of DNA to make two is known as ***replication***.
Strict base-pairing rules are followed. Adenine will pair only with thymine (an A-T pair) and cytosine with guanine (a C-G pair). Each offspring cell receives one old and one new DNA strand. The cell's adherence to these base-pairing rules ensures that the new strand is an exact copy of the old one, minimizing errors (mutations) that may effect the organism.

2.2.2 RNA

RNA is a chemical similar to a single strand of DNA. In RNA, the letter U, which stands for uracil, is substituted for T in the genetic code. RNA delivers DNA's genetic message to the cytoplasm of a cell where proteins are made.

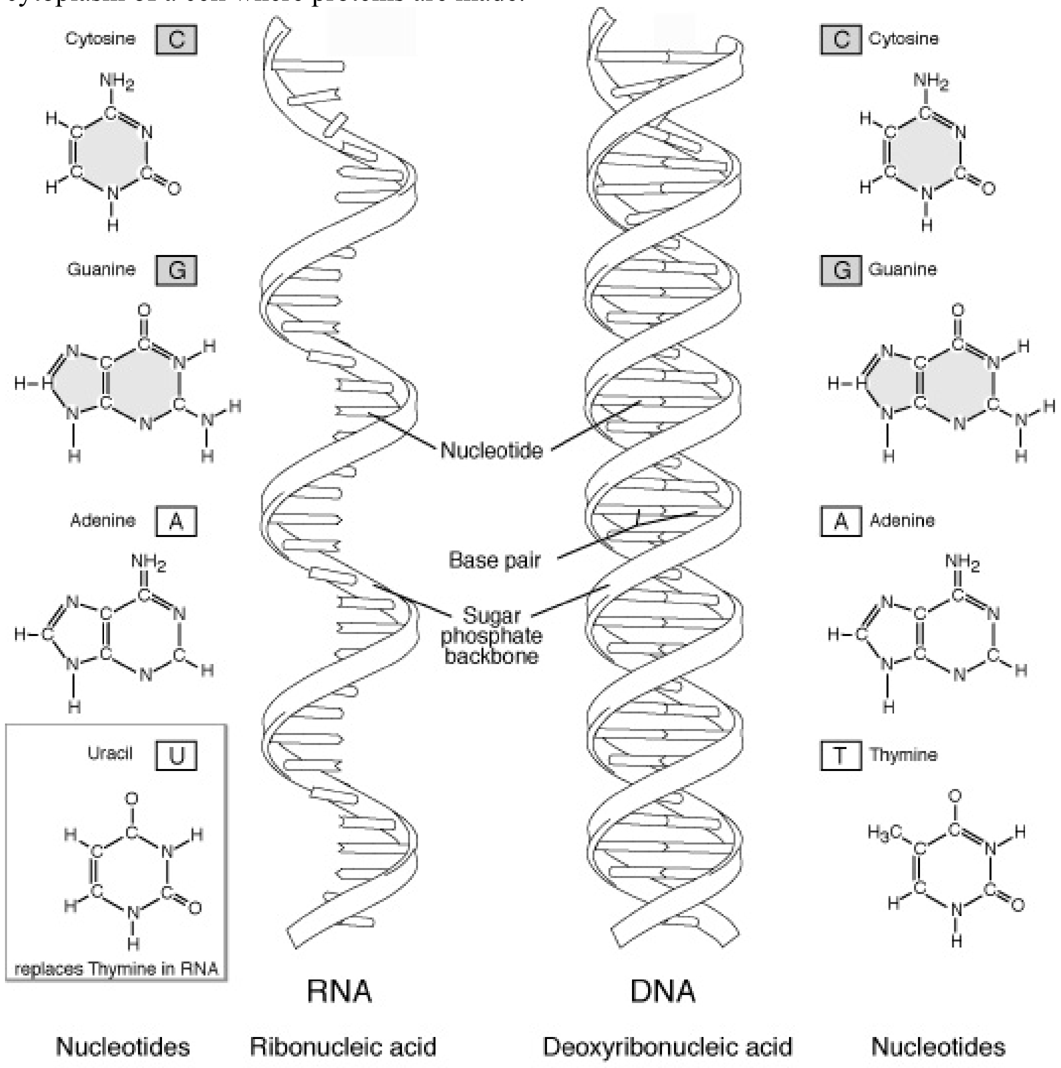


Figure 6 The structure of RNA

With an exception, RNA is the same as DNA. This exception is that RNA replaces the nucleotide thymine (T) with that of uracil (U) so the four nucleotides are adenine (A) which pairs only with uracil (an A-U pair) and cytosine (C) with guanine (a C-G pair). So when the strand of DNA, say T-A-T-C-T-G-T… makes RNA, it can lead to only one arrangement of nucleotides, which is; A-U-A-G-A-C-A... This new RNA, called messenger RNA (mRNA), looks similar to the complementary strand of the original DNA. The difference being that there is a U in the second position instead of a T.
The mRNA has the job of making protein. To do this, it moves out into the cytoplasm of the cell toward the ribosome, which is a tiny intracellular structure used for protein assembly. The ribosome reads off the triplets of the nucleotides along the mRNA chain to determine which amino acid is called for. Then it directs a different class of RNA, a biological retriever called transfer RNA (tRNA) to fetch the right amino acid. The first triplet in the above mRNA example, A-U-A, is the amino acid called isoleucine, so tRNA hunts out isoleucine from amongst the numbers of amino acids suspended in the cell body. Then it brings it to the right spot on the ribosome and goes off to fetch amino acid number two, which in this case is aspartic acid coded as GAC. With the tRNA doing the retrieving and the ribosome doing the assembly, the mRNA chain thus gives the information on which amino acids should be retrieved and eventually a protein chain is formed.
The upshot of it all is that DNA makes RNA and RNA makes protein. However, it doesn't always occur this way as shown by the retroviruses, such as HIV, that begins as a single strand of RNA and directs the cell's nucleus to manufacture DNA from its RNA (thus the use of the word retro which is Latin for backwards).

2.2.3 Genes

A gene can be defined as the functional and physical unit of heredity passed from parent to offspring. Genes are pieces of DNA. A gene is a specific sequence of nucleotide bases, whose sequences carry the information required for constructing proteins, that provide the structural components of cells and tissues as well as enzymes for essential biochemical reactions. The human genome is estimated to comprise at least 100,000 genes. Very roughly, one can think of a gene as encoding a *trait*, such as eye color. Each gene is located at a particular *locus* (position) on the chromosome. Figure 7 shows the construction of a gene.
Human genes vary widely in length, often extending over thousands of bases, but only about 10% of the genome is known to include the protein coding sequences (exons) of genes. Interspersed within many genes are intron sequences, which have no coding function.
For example, the base sequence ATG codes for the amino acid methionine. Since 3 bases code for one amino acid, the protein coded by an average-sized gene (3000 bp) will contain 1000 amino acids. The genetic code is thus a series of codons that specify which amino acids are required to make up specific proteins.
The protein-coding instructions from the genes are transmitted indirectly through ***messenger ribonucleic acid*** (mRNA), a transient intermediary molecule similar to a single strand of DNA. For the information within a gene to be expressed, a complementary RNA strand is produced (a process known as ***transcription***) from the DNA template in the nucleus. This mRNA is moved from the nucleus to the cellular cytoplasm, where it serves as the template for protein synthesis. The cell's protein–synthesizing machinery then translates the codons into a string of amino acids that will constitute the protein molecule for which it codes (Figure 8). In the laboratory, the mRNA molecule can be isolated and used as a template to synthesize a complementary DNA (cDNA) strand, which can then be used to locate the corresponding genes on a chromosome map.
There are more than 200 different specialized cell types in a typical vertebrate. Some are large, some small. For example, a single nerve cell connects your foot to your spinal cord, and a drop of

blood has more than 10,000 cells in it. Some divide rapidly, others do not divide at all; bone marrow cells divide every few hours, and adult nerve cells can live 100 years without dividing. Once differentiated, a cell cannot change from one type to another. Yet despite all of this variation, all of the cells in a multi-cellular organism have exactly the same genetic code. The differences between them come from differences in *gene expression*, that is, whether or not the product a gene codes for is produced, and how much is produced. Control of gene expression is an elaborate dance with many participants. Thousands of biological substances bind to DNA, or bind to other bio-molecules that bind to DNA. Genes code for products that turn on and off other genes, which in turn regulate other genes, and so on.

2.2.4 Chromosomes

Chromosomes are one of the threadlike "packages" of genes and other DNA in the nucleus of a cell. Different kinds of organisms have different numbers of chromosomes. The 3 billion base pairs in the human genome are organized into 24 distinct, physically separate microscopic units called chromosomes. All genes are arranged linearly along the chromosomes.

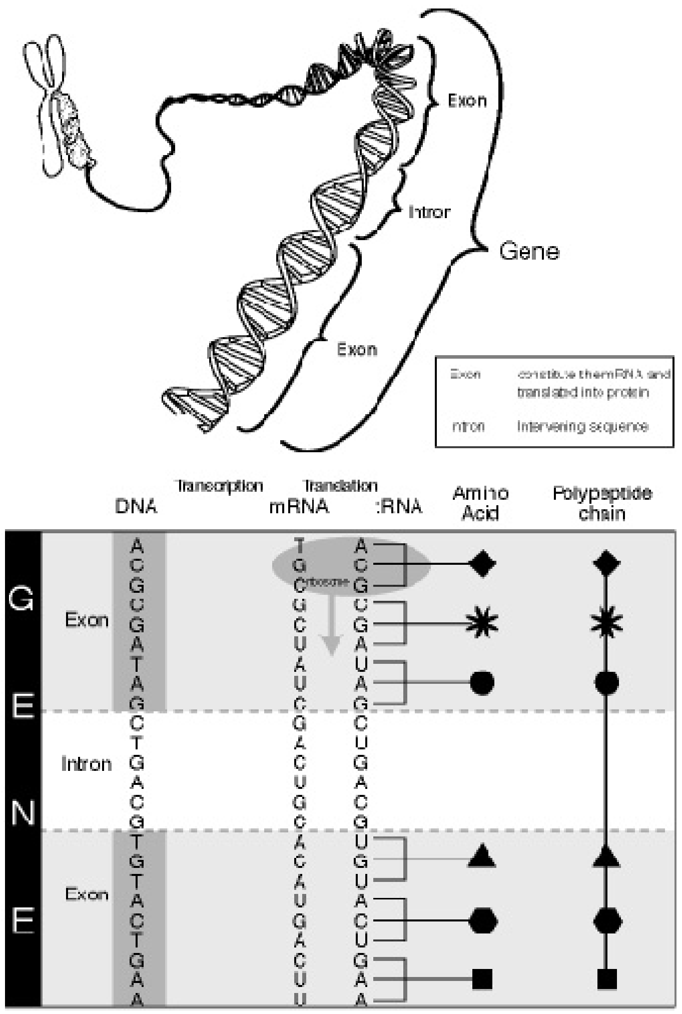


Figure 7 Construction of a gene.

The nucleus of most human cells contains 2 sets of chromosomes. Each set given by each parent. Each set has 23 single chromosomes, that is 22 chromosomes and an X or Y sex chromosome. (A normal female will have a pair of X-chromosomes, whereas a male will have an X and Y pair). Chromosomes contain roughly equal parts of protein and DNA. Chromosomal DNA contains an average of 150 million bases. DNA molecules are among the largest molecules now known. Chromosomes can be seen under a light microscope and when stained with certain dyes, reveal a pattern of light and dark bands. These bands reflecting regional variations in the amounts of A and T versus G and C. Differences in size and banding pattern allow the 24 chromosomes to be distinguished from each other, an analysis called a karyotype. A few types of major chromosomal abnormalities, including missing or extra copies of a chromosome or gross breaks and rejoining (translocations) can be detected by microscopic examination diagnosed by karyotype analysis. Down's syndrome, in which an individual's cells contain a third copy of chromosome 21, is diagnosed by karyotype analysis. Most changes in DNA, however, are too subtle to be detected by this technique and require molecular analysis. These subtle DNA abnormalities (mutations) are responsible for many inherited diseases such as cystic fibrosis and sickle cell anemia or may predispose an individual to cancer, major psychiatric illnesses, and other complex diseases.

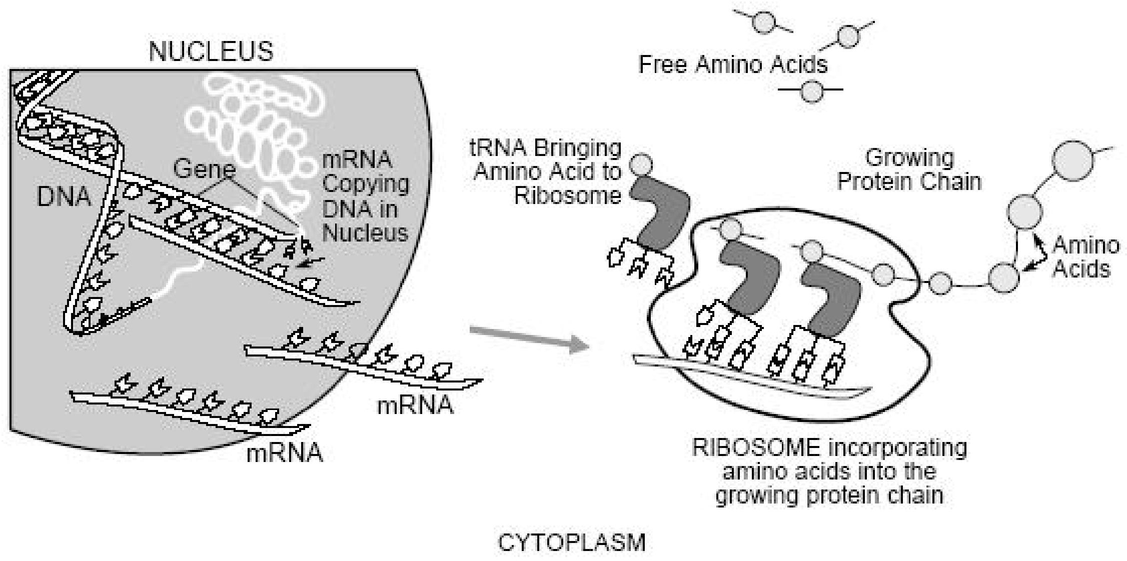

NUCLEUS
Free Amino Acids
tRNA Bringing Amino Acid to Ribosome
Growing Protein Chain
Gene
mRNA Copying DNA in Nucleus
DNA
Amino Acids
mRNA
mRNA
RIBOSOME incorporating amino acids into the growing protein chain
CYTOPLASM

Figure 8 Gene Expression

When genes are expressed, the genetic information (base sequences) on DNA is first transcribed (copied) to a molecule of messenger RNA in a process similar to DNA replication. The mRNA molecules then leave the cell nucleus and enter the cytoplasm, where triplets of bases (codons) forming the genetic code specify the particular amino acids that make up an individual protein. This process, called ***translation****, is accomplished by ribosomes (cellular components composed of proteins and another class of RNA) that read the genetic code from the mRNA, and transfer RNAs (tRNAs) that transport amino acids to the ribosomes for attachment to the growing protein.*

Figure 9 shows a karyotype and Figure 10 illustrates a chromosome.

A chromosome can be conceptually divided into *genes*— each of which encodes a particular protein. The different possible "settings" for a trait (e.g., blue, brown, hazel) are called *alleles*. An allele is one of the variant forms of a gene at a particular *locus* (position or location) on a chromosome. Different alleles produce variation in inherited characteristics such as hair color or blood type. In an individual, one form of the allele (the dominant one) may be expressed more than another form (the recessive one).
Many organisms have multiple chromosomes in each cell. The complete collection of genetic material (all chromosomes taken together) is called the organism's *genome*.
The term *genotype* refers to the particular set of genes contained in a genome. Two individuals that have identical genomes are said to have the same genotype. The genotype is the genetic

identity of an individual that does not show as outward characteristics. The genotype gives rise, under foetal and later development, to the organism's *phenotype*—the observable traits or characteristics of an organism, for example hair color, weight, or the presence or absence of a disease. Phenotypic traits are not necessarily genetic.

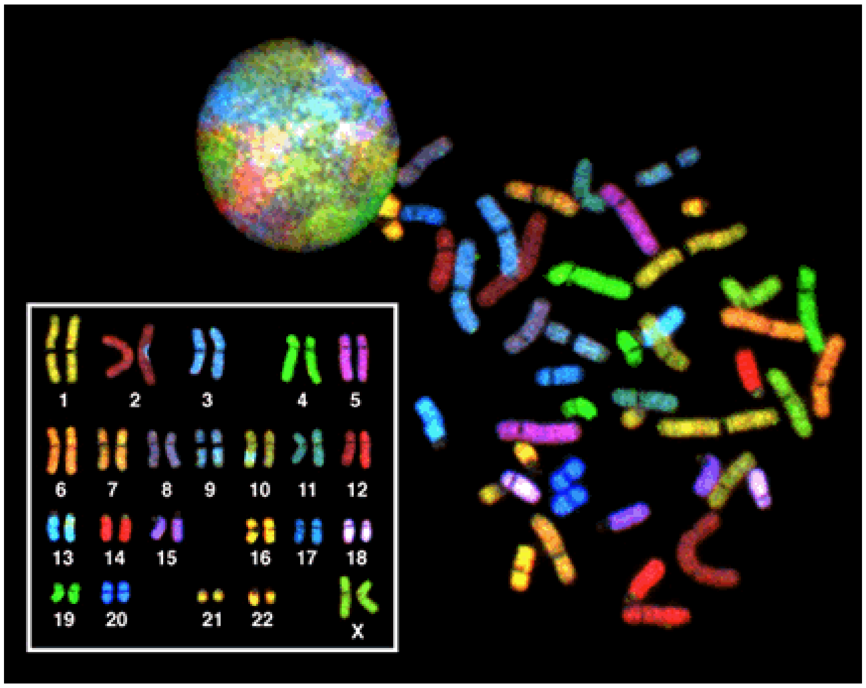


Figure 9 Spectral Karyotype (SKY)
Microscopic examination of chromosome size and banding patterns allows medical laboratories to identify and arrange each of the 24 different chromosomes (22 pairs of autosomes and one pair of sex chromosomes) into a karyotype*, which then serves as a tool in the diagnosis of genetic diseases. A visualization of all of an organism's chromosomes together, each labeled with a different colour is a technique that is useful for identifying chromosome abnormalities.*

Organisms whose chromosomes are arrayed in pairs are called *diploid*. Diploid can be defined as the number of chromosomes in most cells except the gametes (Mature male or female reproductive cell [sperm or ovum] with a haploid set of chromosomes). In humans, the diploid number is 46. Organisms whose chromosomes are unpaired are called *haploid*. Similarly, this can be defined as the number of chromosomes in a sperm or egg cell, that is, half the diploid number. In nature, most sexually reproducing species are diploid, including human beings, who each have 23 pairs of chromosomes in each somatic (all body cells, except the reproductive cells) cell in the body. During sexual reproduction, *recombination* (or *crossover*) occurs: in each parent, genes are exchanged between each pair of chromosomes to form a *gamete* (a single chromosome), and then gametes from the two parents pair up to create a full set of diploid chromosomes. In haploid sexual reproduction, genes are exchanged between the two parents' single□strand chromosomes. Offspring are subject to *mutation*, in which single nucleotides (elementary bits of DNA) are changed from parent to offspring, the changes often resulting from copying errors. The *fitness* of an organism is typically defined as the probability that the organism will live to reproduce (*viability*) or as a function of the number of offspring the organism has (*fertility*).
In genetic algorithms, the term *chromosome* typically refers to a candidate solution to a problem, often encoded as a bit string. The "genes" are either single bits or short blocks of adjacent bits that encode a particular element of the candidate solution (e.g., in the context of multiparameter function optimization the bits encoding a particular parameter might be considered to be a gene). An allele in a bit string is either 0 or 1; for larger alphabets more alleles are possible at each

locus. Crossover typically consists of exchanging genetic material between two single chromosome haploid parents. Mutation consists of flipping the bit at a randomly chosen locus (or, for larger alphabets, replacing the symbol at a randomly chosen locus with a randomly chosen new symbol).

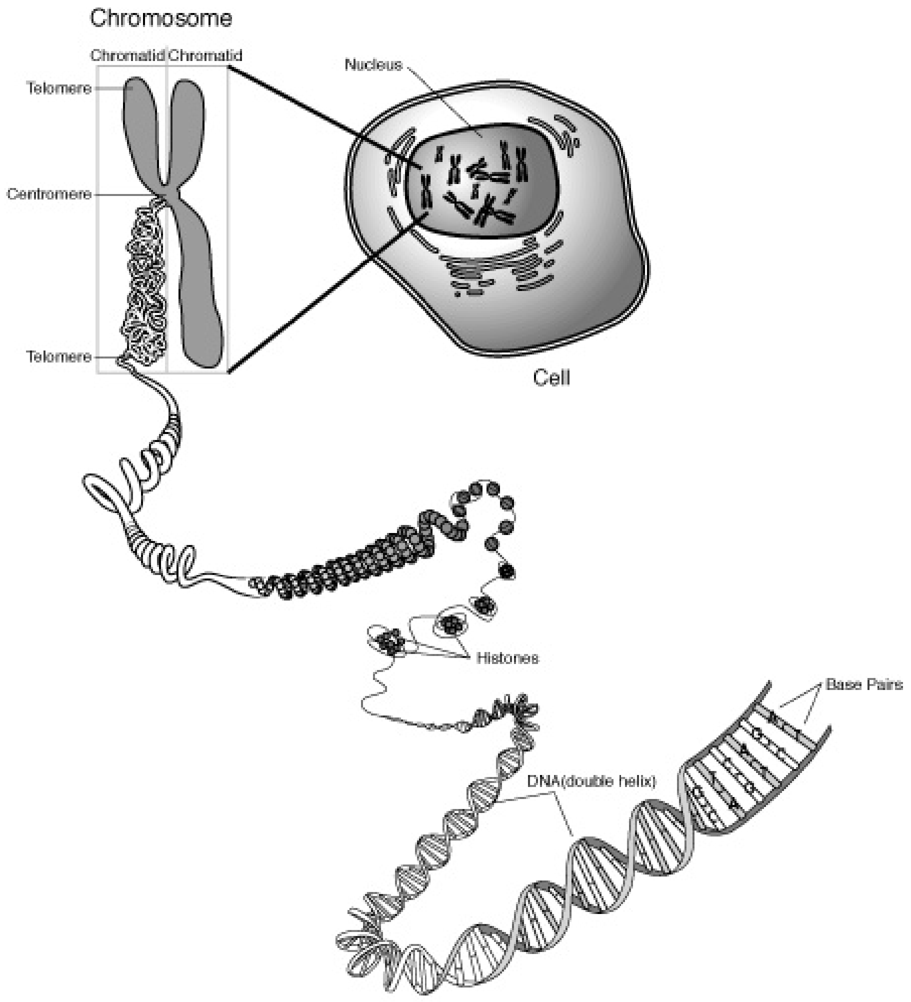


Figure 10 A Chromosome

Most applications of genetic algorithms employ haploid individuals, particularly, single-chromosome individuals. The genotype of an individual in a GA using bit strings is simply the configuration of bits in that individual's chromosome. Often there is no notion of "phenotype" in the context of GAs, although more recently many workers have experimented with GAs in which there is both a genotypic level and a phenotypic level (e.g., the bit-string encoding of a neural network and the neural network itself).

3 Genetic Algorithm basics

3.1 Genetic Algorithms Overview

Genetic Algorithms (GAs) were invented by John Holland in the 1960s and were developed with his students and colleagues at the University of Michigan in the 1970s. Holland's original goal

was to investigate the mechanisms of adaptation in nature and to develop methods in which these mechanisms could be imported into computer systems.
Holland's 1975 book *Adaptation in Natural and Artificial Systems* [4][5] presented the genetic algorithm as an adaptation of biological evolution. His GA is a method for deriving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") a new population. This is achieved by employing "natural selection" together with the genetics□inspired operators of recombination (crossover), mutation, and inversion. Each chromosome consists of "genes" (e.g., bits), and each gene is an instance of a particular "allele" (e.g., 0 or 1). The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average those chromosomes that have a higher fitness factor (defined below), produce more offspring than the less fit ones. Crossover swaps subparts of two chromosomes, roughly imitating biological recombination between two single□chromosome ("haploid") organisms; mutation randomly changes the allele values of some locations (locus) in the chromosome; and inversion reverses the order of a contiguous section of the chromosome.
GAs are the intelligent exploitation of a random search.

3.2 Structure of a single population genetic algorithm

A GA has the ability to create an initial population of feasible solutions (or number of individuals) and randomly initializing them at the beginning of a computation. This initial population is then compared against the specifications or criteria and the individuals that are closest to the criteria, that is, those with the highest fitness factor, are then recombined in a way that guides their search to only the most promising areas of the state or search space. Thus, the first/initial generation is produced
Each feasible solution is encoded as a chromosome (string) also called a genotype and each chromosome is given a measure of fitness (fitness factor) via a fitness (evaluation or objective) function. The fitness of a chromosome determines its ability to survive and produce offspring.
A finite fixed population of chromosomes is maintained.
If the optimization criteria are not met, then the creation of a new generation starts. Individuals are selected (parents) according to their fitness for the production of offspring. Parent chromosomes are combined to produce superior offspring chromosomes (crossover) at some crossover point (locus). All offspring will be mutated (altering some genes in a chromosome) with a certain probability. The fitness of the offspring is then computed. The offspring are inserted into the population replacing the parents, producing a new generation. This cycle is performed until the optimization criteria are reached. In some cases, where the parent already has a high fitness factor, it is better not to allow this parent to be discarded when forming a new generation, but to be carried over. Mutation ensures the entire state-space will be searched, (given enough time) and it is an effective way of leading the population out of a local minima trap.
Thus, in summary, the structure of a single population genetic algorithm as shown in Figure 1, is as follows:

[Start] Generate random population of n chromosomes (suitable **solutions** for the problem) that are L-bits long

[Fitness] Evaluate the fitness f(x) of each chromosome x in the population

[New population] Create a new population by repeating the following steps until the new population is complete

3.1 [Elitism] Select the best chromosome or chromosomes to be carried over to the next generation. This should be an option only.

3.2 [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected). Selection can be done "with replacement", meaning that the same chromosome can be selected more than once to become a parent.

3.3 [Crossover] With a crossover probability p_c, cross over the parents, at a randomly chosen point, to form two new offspring (children). If no crossover is performed, an offspring is the exact copy of parents.

3.4 [Mutation] With a mutation probability p_m, mutate two new offspring at each locus (position in chromosome).

3.5 [Accepting] Place new offspring in the new population

4 [Replace] Replace the old generation with the new generated population for a further run of the algorithm

5 [Test] If the end condition is satisfied, stop, and return the best solution in current population

6 [Loop] Go to step 2

Note that each iteration of this process is called a ***generation***. A GA is typically iterated from 50 to 500 or more generations. The entire set of generations is called a ***run***. At the end of a run there are often one or more chromosomes that match the criteria or specification.

3.2.1 GA operators

A basic genetic algorithm comprises three genetic operators.
- Selection
- Crossover
- Mutation.

Starting from an initial population of strings (representing possible solutions), the GA uses these operators to calculate successive generations. First, pairs of individuals of the current population are selected to mate with each other to form the offspring, which then form the next generation.

- **Selection**

This operator selects the chromosome in the population for reproduction. The more fit the chromosome, the higher its probability of being selected for reproduction. Thus, selection is based on the survival-of-the-fittest strategy, but the key idea is to select the better individuals of the population, as in tournament selection, where the participants compete with each other to remain in the population. The most commonly used strategy to select pairs of individuals is the method of roulette-wheel selection, in which every string is assigned a slot in a simulated wheel sized in proportion to the string's relative fitness. This ensures that highly fit strings have a greater probability to be selected to form the next generation through crossover and mutation. After selection of the pairs of parent strings, the crossover operator is applied to each of these pairs.

- **Crossover (single point)**

The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. This operator randomly chooses a locus (a bit position along the two chromosomes) and exchanges the sub-sequences before and after that locus between two chromosomes to create two offspring. For example, the strings **1110** 0001 0011 and 1000 **0110 0111** could be crossed over after the fourth locus in each to produce the two offspring: **1110 0110 0111** and 1000 0001 0011. The crossover operator roughly imitates biological recombination between two haploid (single chromosome) organisms.
An alternative example is where non-binary chromosomes are used

Parent A = a1 a2 a3 a4 | a5 a6 Parent B = b1 b2 b3 b4 | b5 b6

The swapping of genetic material between the two parents on either side of the selected crossover

point, represented by "|", produces the following offspring:
Offspring A' = a1 a2 a3 a4 | b5 b6
Offspring B' = b1 b2 b3 b4 | a5 a6

- **Mutation**

The two individuals (children) resulting from each crossover operation will now be subjected to the mutation operator in the final step to forming the new generation. This operator randomly flips or alters one or more bit values at randomly selected locations in a chromosome. For example, the string 1000 0001 0011 might be mutated in its second position to yield 1**1**00 0001 0011. Mutation can occur at each bit position in a string with some probability and in accordance with its biological equivalent, usually this is very small, for example, 0.001. If 100% mutation occurs, then all of the bits in the chromosome have been inverted.
The mutation operator enhances the ability of the GA to find a near optimal solution to a given problem by maintaining a sufficient level of genetic variety in the population, which is needed to make sure that the entire solution space is used in the search for the best solution. In a sense, it serves as an insurance policy; it helps prevent the loss of genetic material.

3.3 Multipopulation Evolutionary Algorithms

Better results can be obtained on a broader class of problems by introducing many populations, called sub-populations. Similar, to the single population evolutionary algorithm, each subpopulation evolves for a few generations in isolation before one or more individuals are exchanged between the sub-populations. Much like breeding between different regions, the multipopulation evolutionary algorithm can be considered to model the evolution of a species in a way that is more similar to nature than the single population evolutionary algorithm.

3.4 Properties of GAs

3.4.1 Most important parameters

The most important parameters in GAs, which will be considered more in depth later in this tutorial, are:

- **Population Size**

  Determining the size of the population is a crucial factor. Choosing a population size too small increases the risk of converging prematurely to local minima, since the population does not have enough genetic material to sufficiently cover the state space.
  A larger population has a greater chance of finding the global optimum at the expense of more CPU time.
  The population size remains constant from generation to generation.
- **Evaluation Function (objective function)**
- **Crossover Method**
- **Mutation Rate**

3.4.2 Considerations given to the use of GAs

- A robust search technique is required
- "close" to optimal results are required in a "reasonable" amount of time
- parallel processing can be used
- when the fitness function is noisy
- an acceptable solution representation is available
- a good fitness function is available
- it is feasible to evaluate each potential solution
- a near-optimal, but not optimal solution is acceptable.

- the state-space is too large for other methods

3.4.3 Some properties of GAs

- generally good at finding acceptable solutions to a problem reasonably quickly
- free of mathematical derivatives
- no gradient information is required
- free of restrictions on the structure of the evaluation function
- fairly simple to develop
- do not require complex mathematics to execute
- able to vary not only the values, but also the structure of the solution
- get a good set of answers, as opposed to a single optimal answer
- make no assumptions about the problem space
- blind without the fitness function. ***The fitness function drives the population toward better solutions and is the most important part of the algorithm***.
- not guaranteed to find the global optimum solutions
- probability and randomness are essential parts of GA
- can by hybridized with conventional optimization methods
- potential for executing many potential solutions in parallel

3.4.4 Differences between search methods [6]

The most significant differences between the more traditional search and optimization methods and those of evolutionary and genetic algorithms are that these algorithms,

- work with a coded form of the function values (parameter set), rather than with the actual parameters themselves. So, for example, if we want to find the minimum of the objective function $f(x) = x^2 + 5x + 6$, the GA would not deal directly with x or f(x) values, but would work with strings that encode these values. In this case, strings representing the binary x values would be used,
- use a set, or population, of points spread over the search space to conduct a search, not just a single point on the space. This provides GAs with the power to search spaces that contain many local optimum points without being locked into any one of them, in the belief that it is the global optimum point,
- the only information a GA requires is the objective function and corresponding fitness levels to influence the directions of search. Once the GA knows the current measure of "goodness" about a point, it can use this to continue searching for the optimum,
- use probabilistic transition rules, not deterministic ones,
- are generally more straightforward to apply,
- can provide a number of potential solutions to a given problem,
- are inherently parallel. This is one of the most powerful features of genetic algorithms allowing them to deal with a large number of points (strings) simultaneously,
- the application of GA operators causes information from the previous generation to be carried over to the next.

3.5 GA Vocabulary

- **Population –** a collection of potential solutions or chromosomes
- **Phenotype –** Domain-dependent representation of a potential solution, like a floating-point number.
- **Genotype –** Domain-independent representation of a potential solution, like a binary string.

- Individuals in the population, also called structures, strings, or chromosomes E.g. a binary chromosome can be given by 110010110101101 etc.

- **Genes** – Unit or section of genotype or chromosome (for a binary chromosome can be the individual bits comprising the chromosome, e.g. 110010110101101, or the letters in a string AACBWWSSAA) or a contiguous number of these.
  - Arranged in linear succession and also called features, characters, or decoders
- **Loci** – positions along the string or chromosome, e.g. a position 6 the loci is 6
- **Allele** - value at a particular loci (for a binary chromosome it is the bit value) e.g. at loci 6, it is "0" in the chromosome 11001**0**110101101

4 GA components [26][27]

The design of a GA comprises the following components,

- Encoding technique (Chromosome structure)
- Initialization procedure (Creation)
- Evaluation or Fitness Function (Environment) that assigns a figure of merit to each encoded solution
- Genetic Operators (Selection, Recombination or Crossover, Mutation) that allow parents to be selected and offspring to be generated
- Parameter setting (based on Experience)

4.1 A GA example [7]

Figure 11 shows an annotated screen shot of a GA applet that is designed to find the minimum value of a plot. The plot may be pressure, temperature, etc. An animated "Gif" file GA.html demonstrates how this GA applet operates. It also shows the structure of a single population genetic algorithm as given in Figure 2 and as described above in section 3.2. This animation has **been purposely slowed down** so that the various processes can be easily studied. Please be patient when first running this animation and wait for it to start.

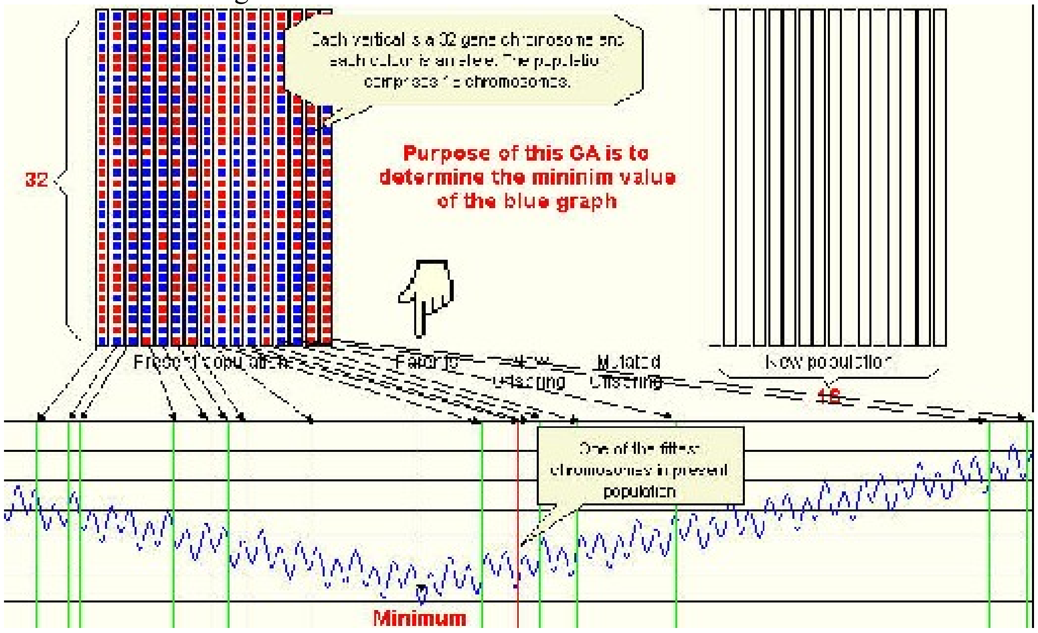

Each vertical is a 32 gene chromosome and each colour is an allele. The population comprises 16 chromosomes.
Purpose of this GA is to determine the mininim value of the blue graph
32
Present population
Parents
New offspring
Mutated offspring
New population
16
One of the fittest chromosomes in present population
Minimum

Figure 11 Annotations explaining the meaning of the various parts of "a GA example" that demonstrates the structure of a GA. The GA is used for determining the minimum value of a graph.

4.2 Encoding

For any GA a chromosome representation is required to describe each individual in the population of interest. The representation scheme determines how the problem is structured in the GA and also determines what genetic operators are used. Each individual or chromosome is made up of a sequence of genes from a certain alphabet. This alphabet could consist of binary digits (0 and 1), floating point numbers, integers, symbols (i.e., A, B, C, D), matrices, etc. In Holland's original design, the alphabet was limited to binary digits. Each element of the string represents a particular feature in the chromosome. The first thing that must be done in any new problem is to generate a code for this problem. How is one to decide on the correct encoding for one's problem? Lawrence Davis, a researcher with much experience applying GAs to real-world problems, strongly advocates ***using whatever encoding is the most natural for your problem, and then devising a GA that can use that encoding*** [36]
For a two dimensional problem, a binary representation of six bits could be chosen, where the three bits on the left represent the x value and the three bits on the right the y value. For example 011101 means: x=011b=3 and y=101b=5. Similarly, if the problem is to minimize a function of three variables, f(x,y,z), each variable may be represented by a 12-bit binary number. The chromosome in this case would therefore contain three genes and consist of 36 binary bits.

One appealing idea is to have the encoding itself adapt so that the GA can make better use of it. Choosing a fixed encoding ahead of time presents a paradox to the potential GA user: for any problem that is hard enough that one would want to use a GA, one doesn't know enough about the problem ahead of time to come up with the best encoding for the GA. In fact, coming up with the best encoding is almost tantamount to solving the problem itself! Thus, most research is currently done by guessing at an appropriate encoding and then trying out a particular version of the GA on it.

4.2.1 Genotypes and Phenotypes revisited

The solution’s representation when undergoing modification, such as the mutation of a binary bit string, is known as the solution's genotype. As mentioned in section 2.1, ***search happens in genotype space***, that is, on chromosomes, but ***selection occurs on phenotypes***, which may be using the decimal floating-point representations of the chromosome values. Thus, the fitness of an individual depends on the performance of the phenotype.
An alternative description of an individual's *phenotype* is the way the solution operates when tested in the problem environment. For example, Figure 12, shows a typical genotypic view (the car design) of a GA population, and Figure 13 shows its equivalent phenotypic view (the built car.)

4.2.2 Unsigned Fixed Point representation

For demonstration purposes, let us characterize the search space with one parameter only. This parameter being "x". We will choose to encode the parameter x using 16 bits, so that we have 16 one-bit genes for each chromosome. Let us assume that we wish to determine the minimum value for the function f(x) in the range x = 0.0, to x = 255.0. Also, we specify that the floating-point value of x is to be to **six** decimal places. The question now is, how do we determine the floating-point value of this 16-bit chromosome parameter x that ranges from 0.0 to 255.0?
We need to determine the floating-point value of x, so that we can use it to determine a fitness value, as will be seen a little later on, and from this permit a selection of the chromosomes with the highest fitness factors.

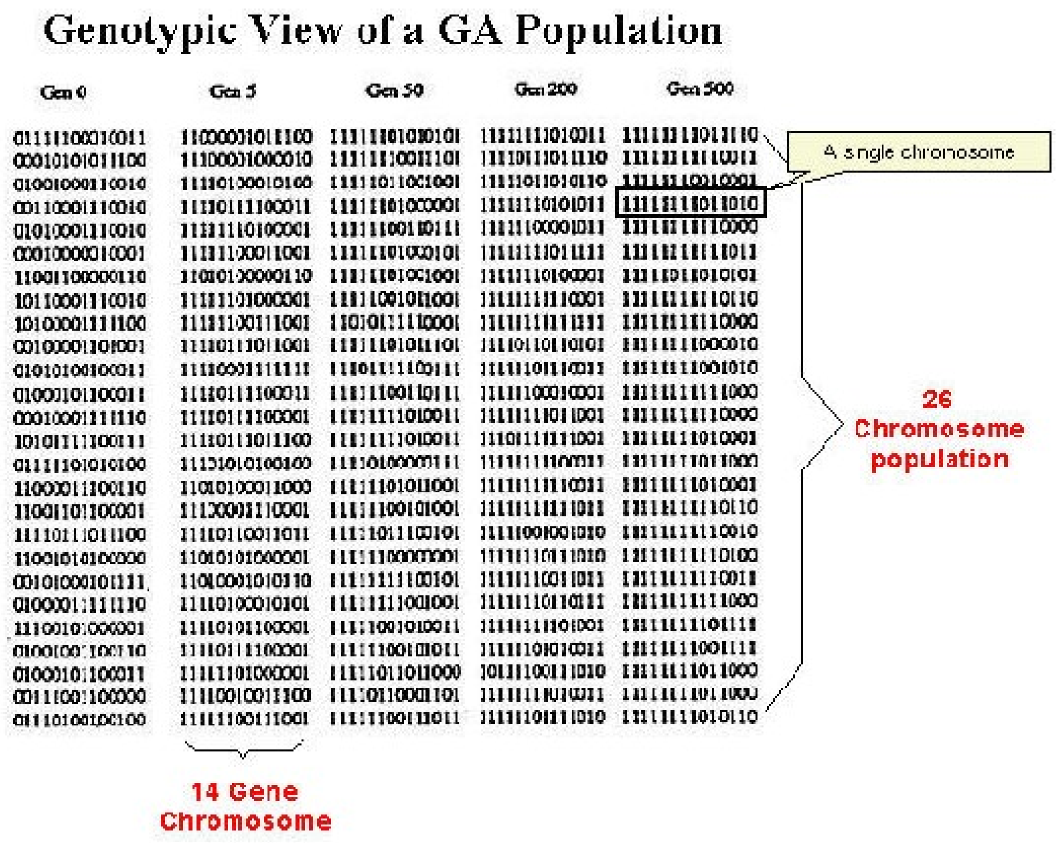


Figure 12 A Genotypic view of a GA population (SEARCH)

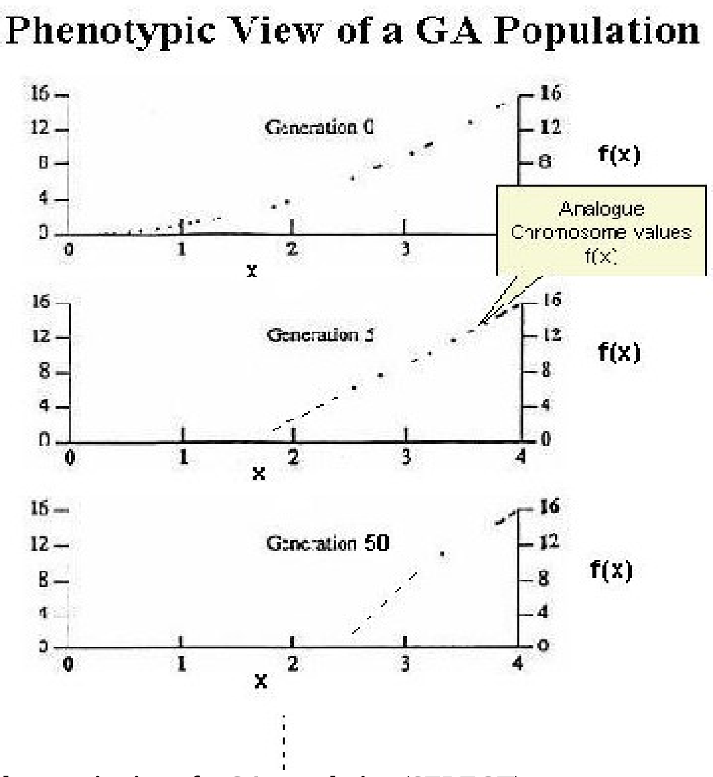


Figure 13 A Phenotypic view of a GA population (SELECT)

Referring to Appendix 1. The $U(a,b)$ representation of a unsigned fixed point number has a integer bits and b fractional bits.
As we require our integer range to be from 0.0 to 255.0, the nearest value of "a", to represent the integer portion of the number, will be 8. This is because 7, produces $2^7 = 128$ which is too small and $2^8 = 256$ which is a little larger than the 255 required. As it was specified that six decimal places was required for the value of x, b = 6, so we get *U(8,6)* and the total number of bits required to represent this range will be 8 + 6 = 14 = N and not 16 as was initially specified. Should we wish to continue using N = 16 bits, then the integer range could increase to 2^{10} = 1024.0 from 255.0

The unsigned fixed point number value x, of a particular N-bit binary number in $U(a,b)$ representation, where

$$N = a + b \tag{1}$$

and x_n is the value of the chromosome at loci n, is given by the expression

$$x = \left(1/2^b\right)\sum_{n=0}^{N-1} 2^n x_n \tag{2}$$

For this example [U(10,6)],

$$x = \left(1/2^b\right)\sum_{n=0}^{N-1} 2^n x_n = \left(1/64\right)\sum_{n=0}^{15} 2^n x_n$$

where x_n represents bit n of the chromosome. The range of a $U(a,b)$ representation is from

$$0 \text{ to } \left(2^N - 1\right)/2^b = 2^a - 2^{-b} \tag{3}$$

For this example, *U(10,6)* the range becomes 0 to $2^{10} - 2^{-6} = 1024.0 - 0.0156 = 1023.9844$ and the 16-bit unsigned fixed-point rational representation *U(10,6)* has the chromosome form

$$b_9 \ldots b_5 b_4 b_3 b_2 b_1 b_0 b_{-1} b_{-2} \ldots b_{-6} = \left(1/2^6\right)\left(b_{15} \ldots b_{11} b_{10} b_9 b_8 b_7 b_6 b_5 b_4 \ldots b_0\right)$$

where bit b_k has a weight of 2^k.

So, given the 16-bit binary number 1001 1000 1110 0011, (that is, $\left(1/2^6\right)\left(b_{15} b_{12} b_{11} b_7 b_6 b_5 b_1 b_0\right)$)
the floating point equivalent would be, $(1/2^6)(2^{15} + 2^{12} + 2^{11} + 2^7 + 2^6 + 2^5 + 2^1 + 2^0)$
= (1/64)(32,768+4,096+2,048+128+64+32+2+1) = 611.546875
Floating-point values do not need to be converted back into binary as the chromosome selections are made in floating point.
In summary, the following subsections describe how to determine the length of the chromosome and the value of the unsigned fixed-point number.

4.2.3 Determining the chromosome length

- Given a range of the unsigned fixed point variable x, as 0 to t, we determine the parameter a, in *U(a,b)* from, $2^a > t$
- From the specification that the unsigned fixed point rational is to be expressed to b decimal places, the value of b in *U(a,b)* is determined
- From equation 1, the number of bits N, in the chromosome is given by N = a + b

- The actual range of the variable x is given by equation 3, to be

$$0 \text{ to } \left(2^N - 1\right)\big/2^b = 2^a - 2^{-b} \qquad (3)$$

4.2.4 Finding the fixed point value from the chromosome

- Given a chromosome of N bits, the unsigned fixed point value x, is found from equation 2 as,

$$x = \left(1/2^b\right)\sum_{n=0}^{N-1} 2^n x_n \qquad (2)$$

where n, is the locus of the gene and x_n, is its *allele* (0 or 1)

4.3 Initialization

Section 3.2 outlined the structure of a single population GA. It starts with the generation of a random population of individuals (chromosomes). The population size remains constant from generation to generation. Determining the size of the population is a crucial factor

- Choosing a population size too small increases the risk of converging prematurely to a local minimum, since the population does not have enough genetic material to sufficiently cover the problem space. Smaller populations tend to become dominated by a single strategy, and lack the robustness required to adapt. This goes back to the old theory that no (biological) species can survive without having a certain minimum population. Population size has another influence on genetic algorithms in that it dilutes the influence of high fitness values on reproductive success. In a population of ten chromosomes, in which one has a fitness of nine and the others a fitness of one, half of all parents will probably be selected from the nine relatively unfit chromosomes, even though the best chromosome is nine times more fit.
- A larger population has a greater chance of finding the global optimum at the expense of more CPU time. For more complicated and difficult the problems (e.g., very small peak in very large search space), use a larger population. Larger populations process more slowly, but they also "cover more ground" and have more schemata to exchange.

The initial population (parental generation) once generated is usually initialized at random. However, if there is information regarding the problem to be solved, then the initial population may be seeded with this information. Population sizes are usually of the order of 50 to 500 in size with no rules for determining the size to use.

4.4 Genetic Operators

To improve chromosomes, two basic types of operators modify the chromosome string. These are; Crossover and mutation. The selection operator determines who will be modified.

4.4.1 Selection

It is useful to distinguish between the *evaluation function* and the *fitness function* used by a genetic algorithm. In this tutorial, the evaluation function, or objective function, provides a measure of performance with respect to a particular set of parameters. The fitness function transforms that measure of performance into an allocation of reproductive opportunities. The evaluation of a string representing a set of parameters is independent of the evaluation of any other string. The fitness of that string, however, is always defined with respect to other members of the current population.
When individuals are modified to produce new individuals, they are said to be *breeding*. Selection determines which individuals are chosen for breeding (recombination) and how many offspring each selected individual produces. The individual (chromosome or string) is first

evaluated by a fitness function to determine the its quality. During testing an individual receives a grade, known as its *fitness*, which indicates how good a solution it is. The period in which the individual is evaluated and assigned a fitness is known as *fitness assessment*. Good chromosomes (those with the highest fitness function) survive and have offspring, while those chromosomes furthest removed or with the lowest fitness function are culled. Constraints on the chromosomes can be modeled by penalties in the fitness function or encoded directly in the chromosomes' data structures.

4.4.1.1 Fitness Assessment

4.4.1.1.1 Deterministic Problems

A fitness function must be devised for each problem to be solved. ***The fitness function and the coding scheme are the most crucial aspects of any GA. They are its core and determine its performance.*** The fitness function must be maximized. In most forms of evolutionary computation, the fitness function returns an individual's assessed fitness as a single real-valued parameter that reflects its success at solving the problem at hand. That is, it is a measure of fitness or a figure-of-merit that is proportional to the "utility" or "ability" of that individual represented by that chromosome. ***This is an entirely user-determined value.***

The general rule to follow when constructing a fitness function is that it should reflect the value of the chromosome in some "real" way. If we are trying to design a power amplifier that produces an output power over a specific audio range, then the fitness function should be the amount of audio power per Hertz, that this amplifier produces, over this specific range.

To prevent premature convergence (the population converging onto a local minimum rather than a global minimum), the population fitness is required to be scaled properly. As the average evaluation of the strings in the population increases, the variance in fitness decreases in the population. There may be little difference between the best and the worst individual in the population after several generations and the selective pressure based on fitness is correspondingly reduced. It is because of this problem that the various methods of fitness scaling are discussed below in this (4.4.1.1) and the following section (4.4.1.2).

In this tutorial, we assign to an individual x_i a *standardized fitness* f_i^s , defined as a fitness parameter in the range (0,1).

The meaning of the chromosome standardized fitness f_i^s ‹ is determined from the following. Let us say that we have the equation of a function g(x). The problem in this case is to determine the minimum of g(x) using only genetic algorithms. To do this we first create a population of chromosomes at random and then we convert these chromosomes into their equivalent floating point numbers. These numbers constitute a set of "x" values that we can now enter into our g(x) to create a set of g(x) values. If we inspect this set of g(x) values we will note that there are some values that are less than others. This is what we are looking for.

If we normalize each g(x) value by dividing it by the sum of all of the g(x) values in the population we produce, for each of the N chromosomes, a number that we will call S_i^s ‹ that is,

$$S_i^s = \frac{g(x_i)}{\sum_{i=0}^{N-1} g(x_i)} \tag{4}$$

Where N is the total number of chromosomes. The range of values of S_i^s is (0,1)

These numbers S_i^s , can now be sorted according to their values. As the smallest S_i^s values are the most important, because we want to find the minimum of the function g(x), we are now able to arrange these numbers is order and assign a fitness value to each one. The smallest number being allocated the highest fitness factor. The fitness factor f_i^a for this case could be,

$$f_i^a = 1 - S_i^S \qquad (5)$$

Where unity would represent the highest fitness factor and 0.0 the lowest.
The ordering of the x floating-point numbers, according to fitness values, allows the ordering of the binary chromosomes. This is in readiness for the next stage, which will be to select pairs for breeding. Those chromosomes whose values of x produce the lowest value of g(x) will have the highest fitness factor.
If a maximum were being sought instead of a minimum, the fitness factors would be given by equation (4) directly.
Let us take a simple example to demonstrate the above. We wish to have,

- Floating point numbers that have three decimal places and whose range is ≈ (0,15). That is, from section 4.2.2, *U(a,b) = U(4,3)* giving the number of bits L, in the chromosome = 7.
- Let us take a random population N, of five chromosomes as shown in Table 1.
- From equation 2, and from the representation of the binary number conversion mapping,

  $$b_3b_2b_1b_0b_{-1}b_{-2}b_{-3} = \left(1/2^3\right)\left(b_6b_5b_4b_3b_2b_1b_0\right)$$

  where bit b_k has a weight of 2^k, we can calculate the x floating point value as shown in Table 1.
- The purpose of this genetic algorithm is to determine the value of x that gives the minimum of the objective function g(x) and also to determine the value of that minimum. In this example we have used only a single variable x, for demonstration purposes. In reality, there would be many variables, thus representing a multidimensional space, in which the GA would be required to find the optimum solutions.
- The function g(x) is given by $g(x) = x^2 - 10x + 26$.

Table 1 Random chromosomes of length L = 7 and population N = 5

Chromosome value (Random)	**Floating point value** $x = \left(1/2^b\right)\sum_{n=0}^{N-1} 2^n x_n$	$g(x)$	**Normalized value** $S_i^s = \frac{g(x_i)}{\sum_{i=0}^{N-1} g(x_i)}$	**Fitness factor** $f_i^a = 1 - S_i^S$	**Chromosome Rank**
1101101	(64+32+8+4+1)/8 = 13.625	75.8571	0.5133	0.4867	5
0101101	(32+8+4+1)/8 = 5.625	1.3906	0.0094	0.9906	1
1001011	(64+8+2+1)/8 = 9.375	20.1406	0.1363	0.8637	3
0010111	(16+4+2+1)/8 = 2.875	5.5156	0.0373	0.9627	2
1011101	(64+16+8+4+1)/8 = 11.625	44.8906	0.3037	0.6963	4
		Total = 147.7945		**Av. = 0.8000**	

Table 1 shows that the most fit chromosome is 0101101 which has a value of x = 1.3906 and which makes the function g(x) = 1.3906. We know from elementary calculus, that the minimum of g(x) is g(x) = 1.0000 when x = 5.000. Our initial run of random chromosomes is to give g(x) = 1.3906.
The population average fitness factor is found to be 4.0000/5.0 = 0.8000 which is quite high as the maximum could only be 1.0000.
If we add a bit more intelligence to the determination of the offspring, the next generation may provide an even closer match. If it does not, then later generations again certainly will.

4.4.1.1.2 Non-Deterministic Problem

Although this tutorial is based on deterministic algorithms, as discussed above in section 4.4.1.1.1, this type of algorithm, will not be suitable for solving all types of problems. In some problems, the solution set may be too large or the best solution is unknown. Optimizing networks, searching large databases and finding optimal paths can often involve solution sets that cannot be practically examined by a deterministic algorithm. In other words, there is no mathematical equation (fitness function) that allows the fitness factor to be determined.

Combinatorial optimization tasks require that solutions be found to problems in involving arrangements of discrete objects. This is quite different from function optimization as different coding, recombination and fitness function techniques are required. The Traveling Salesman Problem (TSP) is a classic problem in game theory that uses the combinatorial optimization approach. The theoretical salesman has several hundred cities that he must visit; his quandary is in ordering his visits such that he will travel the least total distance over his journey. There is a wealth of problems of this genre as may be found in endeavors ranging from the optimization of scheduled delivery routes to the construction of telecommunication alternate routing networks. To encode the travelling salesman problem, say for eight cities, the cities are labeled A through H and a chromosome is a permutation list of the cities representing the salesman's itinerary. For example, the chromosome GFHBACDE would symbolize a journey from G to F to H, etc., ending in city E before return to G. Note the difference in the way the chromosome has been expressed.

The individual letters in the chromosome carry unique meanings (each represents a city); these distinct components of a chromosome can be thought of as *alleles*. Whatever techniques we use for mixing and mutating, each city must be represented once (and only once) in a given chromosome.
A more unusual image processing task is that of producing pictures of criminal suspects where the GA replaces the task usually given to the photo-fit system. The GA generates a number of random faces and the witness selects the two faces that are most similar to the suspect's face. These two face selections are then used to breed more faces for the next generation, The witness acts as the "fitness function" of the GA and is able to control its convergence towards the correct image.

4.4.1.2 Selection and Breeding

Once individuals have had their fitness assessed, they may be selected and bred to form the next generation in the evolution cycle, through repeated application of some selection function. This function usually selects one or two individuals from the old population, copies them, modifies them, and returns the modified copies for addition to the new population. Evolutionary computation is usually done through the application of one or more *breeding operators* to individuals in the population.
Three common operators are *reproduction*, *crossover*, and *mutation*. These, plus three others discussed in this tutorial, are listed in Figure 17.
Crossover, which takes two individuals, called parents, cuts their chromosome strings at some randomly chosen position to produce two "head" and two "tail" segments. The tail segments are then swapped over to produce two new full-length chromosomes. The two offspring inherit some genes from each parent. This is known as single point crossover. Crossover is not usually applied to all parent pairs. A random choice is made where the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, duplicating the parents produces offspring.

As shown in figure 18, individuals are bred using combinations of selection, copying (asexual reproduction —duplicating the individual), recombining (sexual reproduction —mixing and matching two individuals), and modifying.
Mutation is applied to each offspring individually after crossover. It randomly alters a specific gene with a small probability of doing so (typically 0.001). Traditionally, crossover is more important than mutation for rapidly exploring a search space. Mutation provides a small amount of random search and helps ensure that no point in the search space has a zero probability of being searched

4.4.1.2.1 Convergence

With a correctly designed and implemented GA, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum. Convergence is the progression towards increasing uniformity. A gene is said to have converged when 95% of the population share the same value. The population is said to have converged when all of the genes have converged.
At the start of a run, the values for each gene for different members of the population are randomly distributed giving a wide spread of individual fitnesses. As the run progresses some gene values begin to predominate. As the population converges the range of fitnesses in the population reduces. This reduced range often leads to premature convergence and slow finishing.
Premature convergence
A standard problem with GAs is where the genes from a small number of highly fit, but not optimal, chromosomes may tend to dominate the population causing it to converge on a local minimum rather than search for a global minimum. Once the population has reduced its range of fitnesses due to this convergence, the ability of the GA to continue to search for better solutions is effectively prevented. Crossovers of chromosomes that are almost identical produce offspring chromosomes that are almost identical to their parents. The only saving grace is mutation that allows a slower, wider search of the search space to be made.
The schema theorem states that we should allocate reproductive opportunities to individuals in proportion to their relative fitness. However, this allows premature convergence to occur; because the population is not infinite. In order to make GAs work effectively on finite populations the selection process of parents must be modified. Ways of doing this are presented in the next section. The basic idea is to control the number of reproductive opportunities each individual gets, so that it is neither too large, nor too small. The effect is to compress the range of fitnesses and prevent any "super-fit" individuals from having the opportunity to take control.
Slow finishing
If you do not get premature convergence, you probably are going to suffer from slow finishing. After many generations, the population would have converged but can't yet find the global maximum. The average fitness will be high and the range of fitness levels quite small. This means that there is very little gradient in the fitness function. Because of this slight slope, the population slowly edges towards the global maximum rather than going to it quickly.
The same techniques, which will be discussed below, that are used to combat premature convergence also are used to combat slow finishing. That is, they expand the effective range of fitnesses in the population. As with premature convergence, fitness scaling can be prone to over-compression or alternatively expressed, under-expansion due to one "super-poor" individual.

In order to breed individuals, parents must first be selected from among the population according to their fitness. There are several common selection strategies in use:

4.4.1.2.2 High-level approaches used in EC

There are two high-level approaches that are commonly used in evolutionary computation (EC). The traditional approach is known as *generational EC* and the other is *steady-state EC*.

Generational Evolutionary Computation

Generational EC is straightforward. Initially, a population of individuals is created at random. Then the fitness of the individuals in the population is assessed. The better individuals are selected to breed, forming a new population. The new population replaces the original population, and the process repeats at the fitness assessment step. The cycle continues until an ideal individual is discovered or resources are exhausted. The algorithm returns the highest-fit individual it discovered during its run. The generation gap is defined as the proportion of individuals in the population that are replaced in each generation. Most work has used a generation gap of 1, that is, the whole population is replaced in each generation.

Steady-State Evolutionary Computation

Another popular approach, Steady-State EC, requires the same three user-provided functions as the generational approach. However, steady-state EC breeds and replaces only a few individuals at a time in a population, never replacing the whole population in one fell swoop. This permits children to compete directly with parents. This may be a better model of what happens in nature. In short-lived species, parents lay eggs and then die before their offspring hatch. But in longer lived species, offspring and parents live concurrently. The essential difference between a conventional generational replacement GA and a steady state GA is that population statistics, such as average fitness, are recomputed after each mating in a steady state GA and the new offspring are immediately available for reproduction. This gives the opportunity to exploit a promising individual as soon as it is created. Goldberg & Deb's investigations [25] found no evidence that steady-state replacement is fundamentally better than generational.

Other Approaches

The algorithms described above share two constraints in common. First, there is only one population evolved at a time. Second, individuals are assessed independently of one another.
One way to loosen these constraints is to use individuals as part of the assessment of other individuals. For example, to evolve checker players, individuals in the population might receive a fitness assessment based on how well they perform, not against some benchmark standard, but against other individuals in the population acting as opponents. Hopefully, as the population improves, the opponents gradually grow more difficult.
Loosening this constraint result in a family of evolutionary computation algorithms called *competitive fitness* [20].
Another loosening of constraints is to evolve more than one population at a time. This is a common approach to parallelizing evolutionary computation. The popular way to do this is known as the *island model* [21], where evolution occurs in multiple parallel subpopulations, known as *demes*, a notion borrowed from biology [22]. In the island model, demes evolve mostly separately, with occasional "migrations" of highly fit individuals between the demes.
A final common use for multiple populations is a specific form of competitive fitness known as *coevolution*. In coevolution, individuals in populations are assessed through competition against other populations, but breeding is restricted to within each population. One application of coevolution might be to evolve predators and prey. One population of predators is assessed based on their success at hunting down a second population of prey. The prey in turn is assessed based on its success at avoiding the predators. See [20] for a thorough review of coevolution techniques.

4.4.1.2.3 Parent Selection – Mating Pool

Parent selection is the task of allocating reproductive opportunities to each chromosome. In this case, individuals are taken from the population and copied into what is known as a "mating pool". Highly fit individuals are more likely to be copied more than once into this pool and unfit

individuals not at all. In a strict generational replacement scheme, as is discussed in this tutorial, the size of the mating pool is always equal to the size of the population. After this stage, pairs of individuals are taken out of the mating pool at random and mated. This process is repeated until the mating pool is exhausted.
The behavior of the GA depends very much on how individuals are chosen to go into the mating pool. There are two methods used to make this selection.

- Explicit fitness remapping takes the fitness score of each individual and maps it onto a new scale. This remapped value is then used as the number of copies to go into the mating pool (the number of reproductive trials).
- Implicit fitness remapping achieves a similar effect as the first but without computing a modified fitness and mapping it onto a new scale.

4.4.1.2.3.1 Explicit fitness remapping

The average of the number of reproductive trials allocated per individual is unity in order to keep the mating pool the same size as the original population. If each individual's fitness is remapped by dividing it by the average fitness of the population, this effect is achieved. However, this may not be an integer. Since only an integer number of copies of each individual can be placed into the mating pool, there needs to be some way of converting the remapped fitness to an integer without introducing bias. A method known as stochastic universal sampling [10] was devised. This is described below in section 4.4.1.2.6. It is important not to confuse the ***sampling*** method with the ***parent selection*** method. Different parent selection methods may have advantages in different applications, but a good sampling method, such as Baker's, is always good for all selection methods in all applications.
Because we do not want to allocate trials to individuals in direct proportion to raw fitness, many alternative methods for remapping raw fitness in order to prevent premature convergence have been suggested. The major ones are described below.
Fitness scaling
This method is in common use. The maximum number of reproductive trials allocated to an individual is set to a certain value, typically 2.0. This is achieved by subtracting a suitable value from the raw fitness score, then dividing by the average of the adjusted fitness values. Subtracting a fixed amount increases the ratio of maximum fitness to average fitness. Care must be taken to prevent negative fitness values being generated.

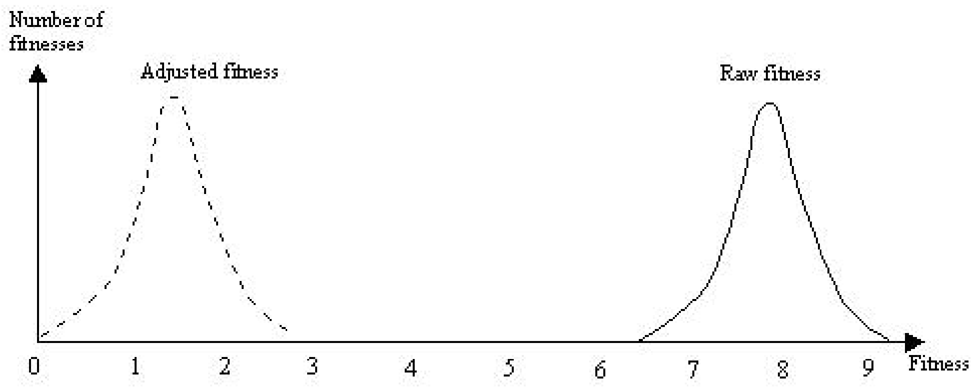


Figure 14 Histograms of raw and adjusted fitness

Figure 14 shows a histogram of raw fitness values with an average fitness of 7.9 and a maximum fitness of 9.2. This gives a maximum to average ratio of 1.16. Thus, without scaling the most fit individual would be expected to receive 1.16 reproductive trials. To apply fitness scaling, we subtract (2 x average – maximum) = (2 x 7.9 – 9.2) = 6.6 from all fitnesses. This gives a

histogram of adjusted fitnesses with an average of 1.3 and a maximum of 2.6, so the maximum to average ratio is now 2.
Fitness scaling compresses the range of fitnesses at the start of a run, thus slowing down convergence and increasing the amount of exploration of the search space.
However, as usual, there is a compromise. The presence of one super fit chromosome, for example, who is super fit by a factor of 10 times the best of the remainder of the population can lead to over compression. Taking the above example, where the compression ratio (maximum to average ratio) is 2, then the remaining population will have fitnesses clustered closely about unity. Although premature convergence has been prevented, the flattening out of the fitness function has produced *genetic drift*, which may lead not just to slower performance but also to a drift away from the maximum.
Genetic drift
This is where any finite genetic population of size *N*, will converge to a single genotype, even if selection is not applied. The expected number of generations until convergence GEN_e, is surprisingly low.
Let *n* denote the number of bits in the chromosome and *N*, the size of the population, then the expected number of generations until equilibrium is given by,

$$E\left(GEN_e\right) \approx 1.4N\left(0.5\ln\left(n\right)+1\right) \qquad (6)$$

Equation 6 is valid for random mating with recombination but without selection and mutation. Note that the $E(GEN_e)$ scales linearly in *N* and only logarithmically in *n*.
Genetic drift is the reason for the surprising fact that small selection intensities decrease the probability to find the optimum.
Fitness windowing
This is similar to fitness scaling except that the amount to be subtracted is chosen differently. The *minimum* fitness in each generation is recorded and the amount subtracted is the minimum fitness observed during the previous *n* generations where *n* is typically 10. With this scheme the selection pressure (that is, the ratio of maximum to average trials allocated) varies during a run and also from problem to problem. The presence of a super-unfit individual will cause under expansion, while super-fit individuals may still cause premature convergence since they do not influence the degree of scaling applied.
The problem with both fitness scaling and fitness windowing is that the degree of compression is dictated by a single, extreme individual, either the fittest or the most unfit. Performance will suffer if the extreme individual is *exceptionally* extreme.
Fitness ranking
Superior to fitness scaling, fitness ranking overcomes the reliance on an extreme individual. Individuals are sorted in order of raw fitness and then reproductive fitness values are assigned according to rank. This may be done linearly or exponentially. This gives a similar result to fitness scaling, in that the ratio of the maximum to average fitness is normalized to a particular value. However, it also ensures that the remapped fitnesses of intermediate individuals are *regularly spread out*. Because of this, the effect of one or two extreme individuals will be negligible, irrespective of how much greater or less their fitness is than the rest of the population. The number of reproductive trials allocated to, for example, the fifth best individual would always be the same, whatever the raw fitness values of those above (or below). The effect is that over compression ceases to be a problem.

4.4.1.2.3.2 Implicit fitness remapping

Implicit fitness remapping methods fill the mating pool without passing through the intermediate stage of remapping the fitness.

4.4.1.2.4 Fitness-Proportional Selection [8].

This selection method, due to Holland [4], normalizes all the fitnesses in the population. These normalized fitnesses then become the probabilities that their respective individuals will be selected. Fitnesses may be transformed in some way prior to normalization; for example, Koza [1] normalizes the adjusted fitness rather than the standardized fitness. One of the problems with fitness-proportional selection is that it is based directly on the fitness. Assessed fitnesses are rarely an accurate measure of how "good" an individual really is.

4.4.1.2.5 Ranked Selection [9].

Another approach that addresses this issue is to rank individuals by their fitness, and use that ranking to determine selection probability. In *linear ranking* [15][16][23], individuals are first sorted according to their fitness values, with the first individual being the worst and the last individual being the best.
Each individual is then selected with a probability based on some linear function of its sorted rank. This is usually done by assigning to the individual at rank i a probability of selection

$$p_i = \frac{1}{\|P\|}\left(2 - c + (2c-2)\frac{i-1}{\|P\|-1}\right) \qquad (6)$$

Where $\|P\|$ is the size of the population P, and $1<c<2$ is the *selection bias*: higher values of *the selective pressure c,* cause the system to focus more on selecting only the better individuals. The best individual in the population is thus selected with the probability $\frac{c}{\|P\|}$; the worst individual is selected with the probability $\frac{2-c}{\|P\|}$.

Some types of selection, such as the roulette wheel, will have problems when there are large differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses then the other chromosomes will have very little chance of being selected.
Rank selection ranks the population first and then every chromosome receives a fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population).
Rank-based fitness assignment overcomes the scaling problems of the proportional fitness assignment. (Stagnation in the case where the selective pressure is too small or premature convergence where selection has caused the search to narrow down too quickly.) The reproductive range is limited, so that no individuals generate an excessive number of offspring. Ranking introduces a uniform scaling across the population and provides a simple and effective way of controlling selective pressure.

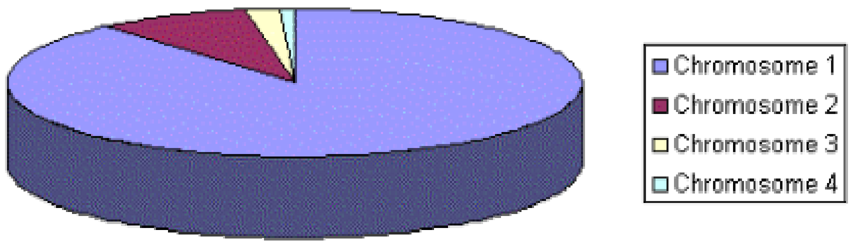


Figure 15 Fitness value distribution before ranking

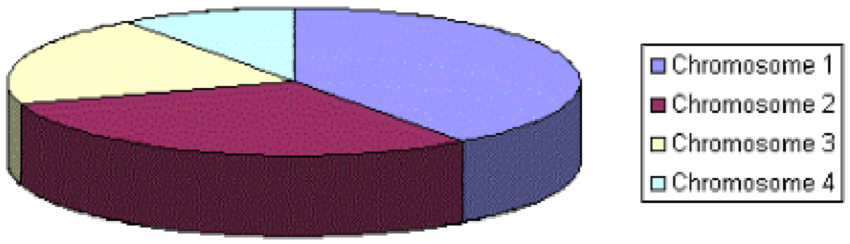


Figure 16 Fitness value distribution after ranking

As shown in following diagram, the situation radically changes after changing the fitness values to the numbers determined by the ranking.
This method can lead to slower convergence, because the best chromosomes do not greatly differ in fitness values from the less fit chromosomes.

4.4.1.2.6 Stochastic universal sampling [10]

Stochastic universal sampling provides zero bias and minimum spread. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in roulette-wheel selection. Here equally spaced pointers are placed over the line as many as there are individuals to be selected. Consider n the number of individuals to be selected, then the distance between the pointers is 1/n and the position of the first pointer is given by a randomly generated number in the range [0, 1/n].
Assuming the following in Table 4,

Table 4 Selection probability and fitness value

Number of individual	fitness value	selection probability
1	2.0	0.18
2	1.8	0.16
3	1.6	0.15
4	1.4	0.13
5	1.2	0.11
6	1.0	0.09
7	0.8	0.07
8	0.6	0.06
9	0.4	0.03
10	0.2	0.02
11	0	0.0

Figure 17 shows the selection for the above example plotted on a line graph.
Taking the length of the line graph as unity, for 9 new individuals to be selected for mating, the distance for equal distances between the pointers is 1/9=0.111.
A single random number drawn in the range of the highest selection probability, that is, [0, 0.18] produces, for example, 0.1. This gives the starting position of pointer 1. From this position, the next pointer (pointer 2) is placed a distance of 0.111, and so on until all pointers have been placed.
Noting where each pointer points determine which individual is selected. After selection the mating population consists of the individuals: 1, 2, 2, 3, 4, 5, 6, 7 and 10.
Stochastic universal sampling provides a selection of offspring that may be lead to faster convergence to the solution to a problem than those formed from parents using the roulette wheel.

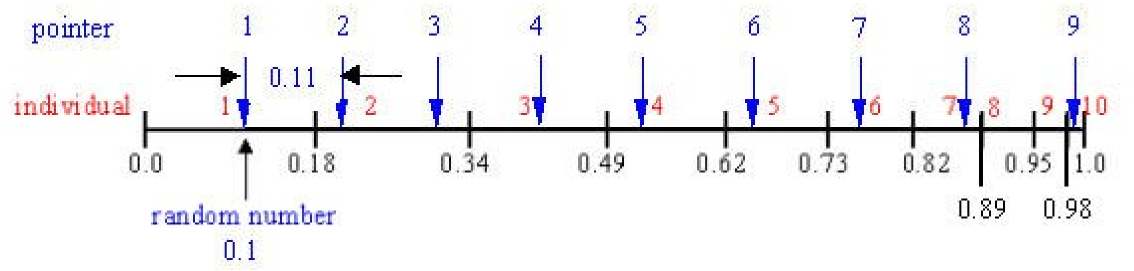


Figure 17 **Stochastic universal sampling**

4.4.1.2.7 Roulette wheel selection [10]

The simplest selection scheme is roulette-wheel selection, also called stochastic sampling with replacement. Picture a standard gambler's roulette wheel—for a genetic algorithm, each slot on the wheel represents a chromosome from the parent generation; the width of each slot represents the relative fitness of a given chromosome. Essentially, the simulated roulette wheel generates a random number that is some fraction of the total fitness of the parent population; then it counts around the wheel until it finds the selected chromosome. The largest fitness values tend to be the most likely resting-places for the marble, since they have larger slots.
Consider an example that has a hypothetical population of five chromosomes as shown in Table 5

Table 5 **Hypothetical Population and its Fitness**

Chromosome	**Fitness**	**% Fitness**
10110110	20	34
10000000	5	8
11101110	15	25
10010011	8	13
10100010	12	20
TOTAL	**60**	

The total fitness of this population is 60, permitting each chromosome to have its fitness expressed as a percentage. Figure 18 shows the relative sizes of pie slices as assigned by fitness. Chromosome 10110110 has a 34% chance of being selected as a parent, whereas chromosome 10000000 has only an 8% chance of generating a new chromosome. Chromosomes selected, by fitness, from the old generation will parent each chromosome in a new generation. Selecting five pairs of parents requires the algorithm to generate ten random numbers, as shown in Table 6. Independently, pairs of random numbers between 0.0 and 1.0 are generated.
Table 6 shows the tabulation of these random numbers.
Where the position of each of these random numbers around the wheel is placed determines a chromosome. For example, random number 0.73 occurs between 0.67 and 0.80 in Figure 18 above. This range belongs to chromosome 10010011. Table 6 is completed in this fashion for the rest of the random numbers giving both the father and mother chromosomes.
The chromosomes with the highest fitness, 10110110 and 11101110 each parent three members of the new population; even the chromosomes with the lowest fitness will be parents once.
Multiple copies of the same string can exist in the mating pool. This is even desirable, since the stronger strings will begin to dominate, eradicating the weaker ones from the population. There are difficulties with this however, as it can lead to premature convergence on a local optimum.

Table 6 Roulette Selecting Parent Chromosomes

Random Numbers	Father Chromosome	Mother Chromosome
0.73, 0.62	10010011	11101110
0.08, 0.53	10110110	11101110
0.82, 0.05	10100010	10110110
0.30, 0.45	10110110	11101110
0.37, 0.80	10000000	10100010

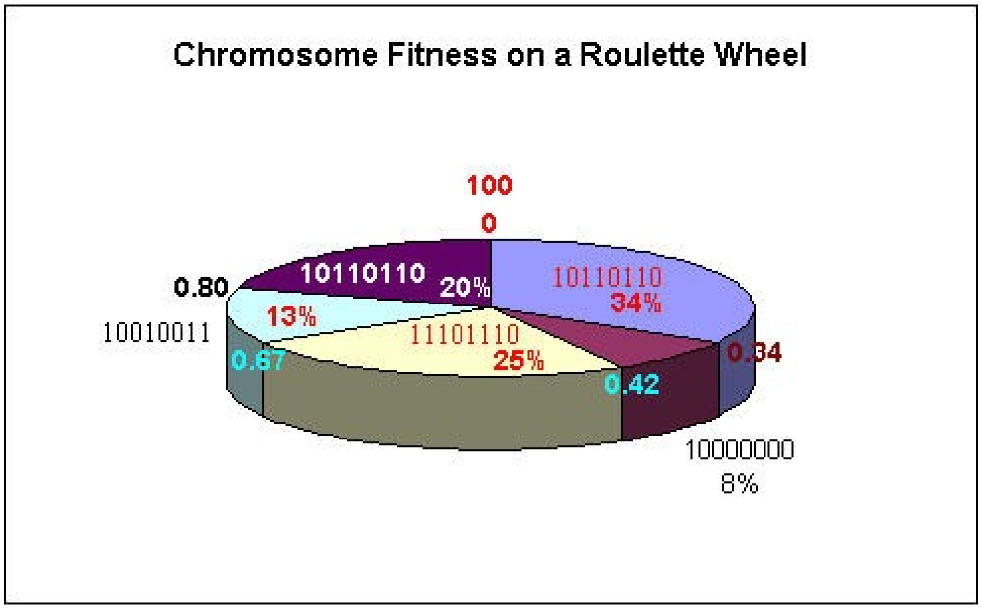


Figure 18 Chromosome fitness on a roulette wheel

4.4.1.2.8 Truncation selection [13][14][24]

Compared to the previous selection methods modeling natural selection, truncation selection is an artificial selection method. Breeders for large populations/mass selection use it.
In truncation selection, **individuals are sorted according to their fitness.** The next generation is formed from breeding only the best individuals in the population. One form of truncation selection, (μ,λ) selection, works as follows. Let the population size $\lambda = k\mu$ where k and μ are positive integers. The μ best individuals in the population are "selected". Each individual in this group is then used to produce k new individuals in the next generation. In a variant form, $(\mu + \lambda)$ selection, μ individuals are "selected" from the union of the population and the μ parents which had created that population previously. The (μ,λ) and $(\mu + \lambda)$ selection are described further in [17].

4.4.1.2.9 Tournament selection [14][25]

This selection mechanism is popular because it is simple, fast, and has well-understood statistical properties. In tournament selection, a pool of n individuals is picked at random from the population. These are independent choices: an individual may be chosen more than once. Then tournament selection selects the individual with the highest fitness in this pool. Clearly, the larger the value n, the more directed this method is at picking highly fit individuals. On the other hand, if $n = 1$, then the method selects individuals totally at random. Popular values for n include 2 and 7. Two is the standard number for genetic algorithm literature, and is not very selective. Seven is used widely in the genetic programming literature, and is relatively highly selective.

The parameter for tournament selection is the tournament size T. The value parameter takes values ranging from 2 - N (number of individuals in population). Table 7 shows the relation between tournament size and selection intensity [14] using equation 7.

Table 7 Relation between tournament size and selection intensity

Tournament size	1	2	3	5	10	30
Selection intensity	0	0.56	0.85	1.15	1.53	2.04

Selection intensity S (expected average fitness value of the population after applying a selection method to the normalized Gaussian distribution)

$$\mathrm{S}(\mathrm{T}) \approx \sqrt{2\left(\ln T - \ln\sqrt{4.14\ln T}\right)} \tag{7}$$

Using larger tournament size has the effect of increasing the selection intensity, since below average individuals are less likely to win a tournament, while above average individuals are more likely to win. High selection intensity leads to premature convergence and thus a poor quality of the solutions.
Loss of diversity L (proportion of individuals of a population that is not selected during the selection phase)

$$\mathrm{L}(\mathrm{T}) = T^{-\left(\frac{1}{T-1}\right)} - T^{-\left(\frac{T}{T-1}\right)} \tag{8}$$

(About 54% of the population is lost at tournament size T = 5).
Selection variance V (expected variance of the fitness distribution of the population after applying a selection method to the normalized Gaussian distribution)

$$\begin{aligned} V(T) &= 1 - 0.096\ln\left(1 + 7.11(T-1)\right) \\ V(4.73) &\approx 1 - \frac{1}{\pi} \end{aligned} \tag{9}$$

4.4.2 Recombination or Crossover

After selection has been carried out recombination can occur.
Crossovers are (sometimes) deterministic operators that capture the best features of two parents and pass it to a new offspring. The population is recombined according to the probability of crossover p_c. When a population has been entirely replaced by children, the new population is known as the next *generation*. The whole process of finding an optimal solution is known as *evolving* a solution.

4.4.2.1 Binary valued recombination

4.4.2.1.1 One-point crossover

The traditional GA uses 1-point crossover, where the two mating chromosomes are each cut once at corresponding points and the selections after the cuts exchanged. This is shown below in figure 19, for the case, where two binary strings 1110 1100 1101 0110 and yyzy zyyz yzyy zzzy are selected as parents. The locus point is randomly chosen.

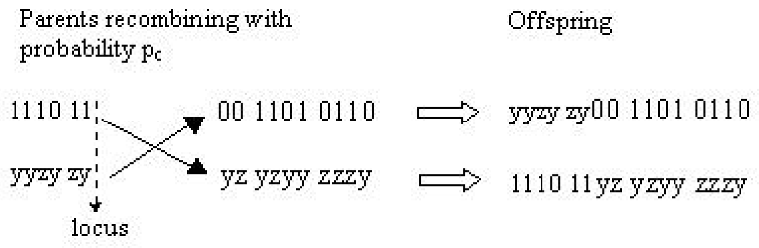


Figure 19 One-point crossover

4.4.2.1.2 Two-point crossover

In two-point crossover chromosomes are regarded as loop formed by joining the ends together. To exchange a segment from one loop with that from another loop requires the selection of two randomly chosen crossover or cut points. Strings exchange the segment that falls between these two points as shown in Figure 20.
DeJong [28] first observed that two-point crossover treats strings as if they form a ring, where each of the beads in the ring represent bits as shown in figure 21. The gene between loci 1 and loci 2 is changed out in both parents to produce the offspring, as shown in figure 20. One-point crossover is a special case of two-point crossover where one of the randomly chosen crossover points always occurs at the wrap-around position between the first and last bit, as shown in fig 21.

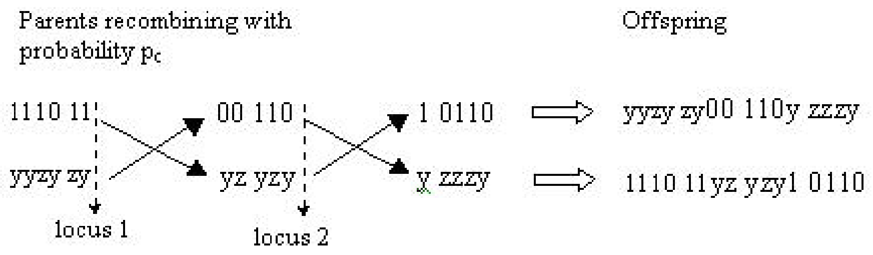


Figure 20 Two-point crossover

It is generally accepted that two-point cross over is better than one-point crossover in finding a speedy solution.

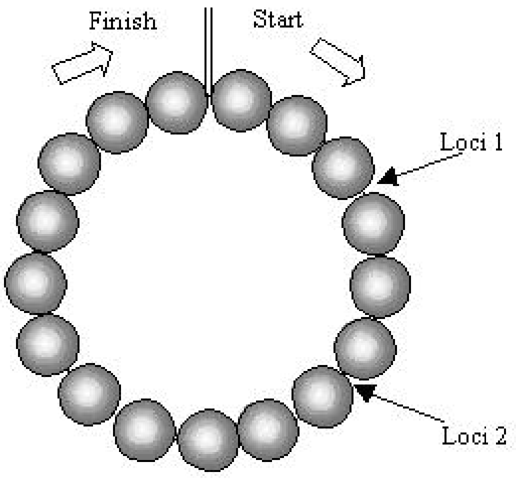


Figure 21 Two-point crossover viewed as a ring

4.4.2.1.3 Uniform crossover

This form of crossover is different from one-point crossover. Copying the corresponding gene from one or the other parent, chosen according to a randomly generated *crossover mask* creates each gene in the offspring. Where there is a "1" in the crossover mask, the gene is copied from the first parent and where there is a "0" in the mask, the gene is copied from the second parent as shown in figure 22. The process is repeated with the parents exchanged to produce the second offspring. A new crossover mask is randomly generated for each pair of parents.
Offspring therefore, contain a mixture of genes from each parent. The number of effective crossing points is not fixed, but will average L/2 where L is the chromosome length.

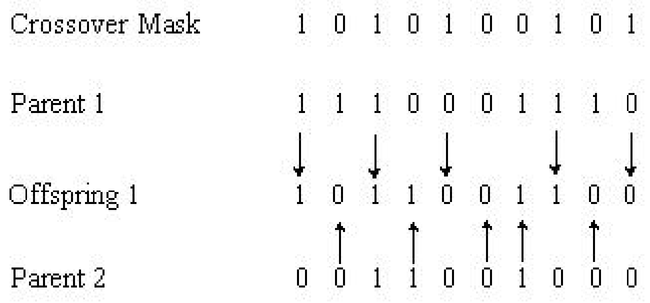

Crossover Mask 1 0 1 0 1 0 0 1 0 1
Parent 1 1 1 1 0 0 0 1 1 1 0
Offspring 1 1 0 1 1 0 0 1 1 0 0
Parent 2 0 0 1 1 0 0 1 0 0 0

Figure 22 Uniform crossover

4.4.2.1.4 Shuffle crossover

Shuffle crossover [29] is related to uniform crossover. A single crossover position (as in single-point crossover) is selected. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshuffled. This removes positional bias as the variables are randomly reassigned each time crossover is performed.

4.4.2.1.5 Partially matched crossover (PMX)

Partially matched crossover (PMX) arose in an attempt to solve the blind Travelling Salesman Problem (TSP). In the blind TSP, fitness is entirely based on the ordering of the cities in a chromosome and as such, we need to maintain valid permutations during reproduction. PMX begins by selecting two points -- two and five, in this case -- for its operation.
Parent chromosome 1: AB|CDE|FGH
Parent chromosome 2: GF|**H**BA|CDE

The PMX algorithm

- notes that the H allele in Chromosome 2 will replace the C allele in Chromosome 1
- so it replaces C with H and H with C for both chromosomes

The same process is accomplished for the other two alleles being swapped, that is,

- B replaced D and D replaces B in both chromosomes
- A replaces E and E replaces A in both chromosomes

and the end result is two offspring with these encodings:
Offspring 1: ED|HBA|FG**C**
Offspring 2: GF|CDE|HBA

Each offspring is a new permutation, shuffling some alleles while preserving a section of a parent organism.

4.4.2.1.6 Order crossover (OX)

Order crossover involves the removal of some alleles and the shifting of others. Given the crossover points and parent chromosomes as in the PMX example, OX would remove the incoming alleles like so (a dash represents a blank allele):
Offspring 1: --|HBA|FG-
Offspring 2: GF|CDE|---

Then, beginning after the second crossover point, OX shifts alleles to the left (wrapping around the end of the chromosome if necessary), filling empty alleles and leaving an opening for the swapped-in section:
Offspring 1: BA|---|FGH
Offspring 2: DE|---|GFC

To finish the process, OX exchanges the alleles within the crossover boundaries, finishing the two offspring.
Offspring 1: BA|CDE|FGH
Offspring 2: DE|HBA|GFC

Where PMX preserves the absolute position of a city allele within chromosomes, OX preserves the order of cities in the permutation.

4.4.2.1.7 Cycle crossover (CX)

This form of crossover works in an entirely different fashion, by swapping a specific set of cities between chromosomes. For comparison, using the same two chromosomes for the example in PMX and OX.
Parent 1: ABCDEFGH
Parent 2: GFHBACDE

In generating offspring, CX begins with the first cities of the two parent chromosomes:
Offspring 1: G-------
Offspring 2: A-------

A search of Parent 1 finds the just-introduced G allele in position 7. Another swap occurs:
Offspring 1: G-----D-
Offspring 2: A-----G-

The search-and-swapping process continues until the allele first replaced in Parent 1 -- the A -- is found in a swap between chromosomes. CX then fills the remaining empty alleles from corresponding elements of the parents. The final offspring look like this:
Offspring 1: GECBAFDH
Offspring 2: ABHDECGE

The inversion operator isn't a form of crossover; it reverses a sequence of alleles. Inversion preserves the nature of a permutation while reordering its elements. Here are two examples of inversion applied to the test chromosomes:
ABC|DEFGH| inverts to ABCHGFED

G|FHB|ACDE inverts to GBHFACDE

Mutation occurs after reproduction. In the case of a permutation chromosome, a mutation changing only one allele would create duplicate alleles and lose others. To overcome this problem, mutation may be defined as the swapping of two alleles within the chromosome. For example, swapping A and E alleles in ABCDEFGH would produce the offspring EBCDAFGH.

4.4.2.1.8 Other crossover techniques

Many other techniques have been suggested. The idea that crossover should be more probable at some string positions (loci) than others has some basis in nature. The general principle is that the GA adaptively learns which loci should be favoured for crossover. This information is recorded in a punctuation string, which is itself part of the chromosome and so is crossed over and passed on to descendants. In this way, punctuation strings that lead to good offspring will themselves be propagated through the population.

4.4.2.2 Real valued recombination

4.4.2.2.1 Discrete recombination [30]

Discrete recombination performs an exchange of variable values between the individuals. Consider the following two individuals with 5 variables each.

individual 1	15	67	98	12	43
individual 2	127	6	108	16	44

For each variable the parent who contributes its variable to the offspring is chosen randomly with equal probability.

sample 1	2	2	1	2	1
sample 2	1	2	1	1	1

After recombination the new individuals are created:

offspring 1	127	6	98	16	43
offspring 2	15	6	98	12	43

Discrete recombination can be used with any kind of variables (binary, real or symbols).

4.4.2.2.2 Intermediate recombination [30]

Intermediate recombination is a method only applicable to real variables (and not binary variables). Here the variable values of the offspring are chosen somewhere around and between the variable values of the parents.
Offspring are produced according to the rule:

$$\text{offspring} = \text{parent } 1 + \alpha\,(\text{parent } 2 - \text{parent } 1) \qquad (10)$$

where α is a scaling factor chosen uniformly at random over an interval $[-d, 1 + d]$. In intermediate recombination $d = 0$, for extended intermediate recombination $d > 0$. A good choice is $d = 0.25$. Each variable in the offspring is the result of combining the variables according to the above expression with a new Alpha chosen for each variable.
Figure 2 shows the area of the variable range of the offspring defined by the variables of the parents.

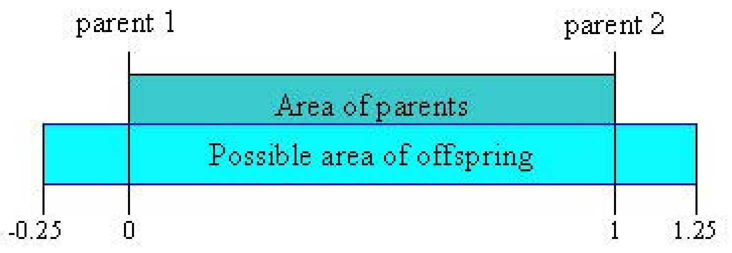


Figure 23 Area for variable value of offspring compared to parents in intermediate recombination

Consider the following two individuals with 5 variables each:

individual 1	15	67	98	12	43
individual 2	127	6	108	16	44

The chosen Alpha for this example are:

sample 1	0.5	1.1	-0.1	0.3	-0.2
sample 2	0.1	0.4	0.7	1.2	0.9

The new individuals are calculated as:

offspring 1	71.0	-0.1	97.0	13.2	42.8
offspring 2	26.2	42.6	105.0	16.8	43.9

4.4.2.2.3 Line recombination [30]

Line recombination is similar to intermediate recombination, except that only one value of α for all variables is used:

individual 1	15	67	98	12	43
individual 2	127	6	108	16	44

The chosen Alpha for this example are:

sample 1	0.5
sample 2	0.1

The new individuals are calculated from equation 10,

offspring 1	71.0	36.5	103.0	14.0	43.5
offspring 2	26.2	60.9	99.0	43.4	43.1

4.4.2.3 Summary

Of all the crossover methods described above, the outcome from various researchers [26][33][34][35] is as follows,

- There is not more than 20% difference in speed among 1-, 2-point, multi-point and uniform crossover. However 8-point crossover maybe considered as a serious contender.
- If the population is small compared with the problem complexity, probably better to use uniform crossover
- If the population is large compared with the problem complexity, probably better to use 2-point crossover

4.4.3 Mutation [30][32]

After recombination offspring undergo mutation. Although it is generally held that crossover is the main force leading to a thorough search of the problem space, mutations are probabilistic background operators that try to re-introduce needed chromosome features (bit or allele) into populations whose features have been inadvertently lost. Mutation can assist by preventing a (small) population prematurely converging onto a local minimum and remaining stuck on this minimum due to a recessive gene that has infected the whole population (genetic drift). It does this by providing a small element of random search in the vicinity of the population when it has largely converged. Crossover alone cannot prevent the population converging on a local minima. Mutation generally finds better solutions than a crossover-only regime although crossover gives much faster evolution than a mutation-only population. As the population converges on a solution, mutation becomes more productive and crossover less productive. Consequently, it is not a choice between crossover and mutation but, rather the balance among crossover, mutation and selection that is important.
Offspring variables are mutated by the addition of small random values (size of the mutation step), with low probability. The probability of mutating a variable $p_{m,}$ is set to be inversely proportional to the number of bits (variables) "n", in the chromosome (dimensions). The more dimensions one individual has the smaller the mutation probability is required to be.

A mutation rate **m = 1/n** produces ***almost optimal results*** for a broad class of test functions where the mutation rate is independent of the size of the population. Varying the mutation rate by increasing it at the beginning of a search and a decreasing it to 1/n at the end as the population converges, gives an insignificant improvement in the search speed.

4.4.3.1 Binary mutation

Although the mutating of real-valued numbers exists, this is not discussed in this tutorial. Further information may be found in references [30] and [31].
For binary valued individuals mutation means the flipping of variable values. For every individual the variable value to change is chosen uniform at random.

Table 8, shows an example of a binary mutation for an individual with 12 variables, variable 3 is mutated.

Table 8 Binary mutation

Before mutation	1	1	0	0	0	1	1	0	1	1	0	0
After mutation	1	1	**1**	0	0	1	1	0	1	1	0	0

Figure 24 shows in diagram form the different breeding methods.

4.4.4 Linkage and Inversion

Linkage

Linkage refers to the phenomenon whereby a set of bits acts as "coadapted alleles" that tend to be inherited together as a group. In this case an *allele* would correspond to a particular bit value at a specific locus on the chromosome. Linkage is sometimes defined by physical adjacency of bits in a string encoding; this implicitly assumes that 1-point crossover is the operator being used. Linkage under 2-point crossover is different and must be defined with respect to distance on the chromosome when treated as a ring. Nevertheless, linkage usually is equated with physical adjacency on a string as measured by *defining length*.

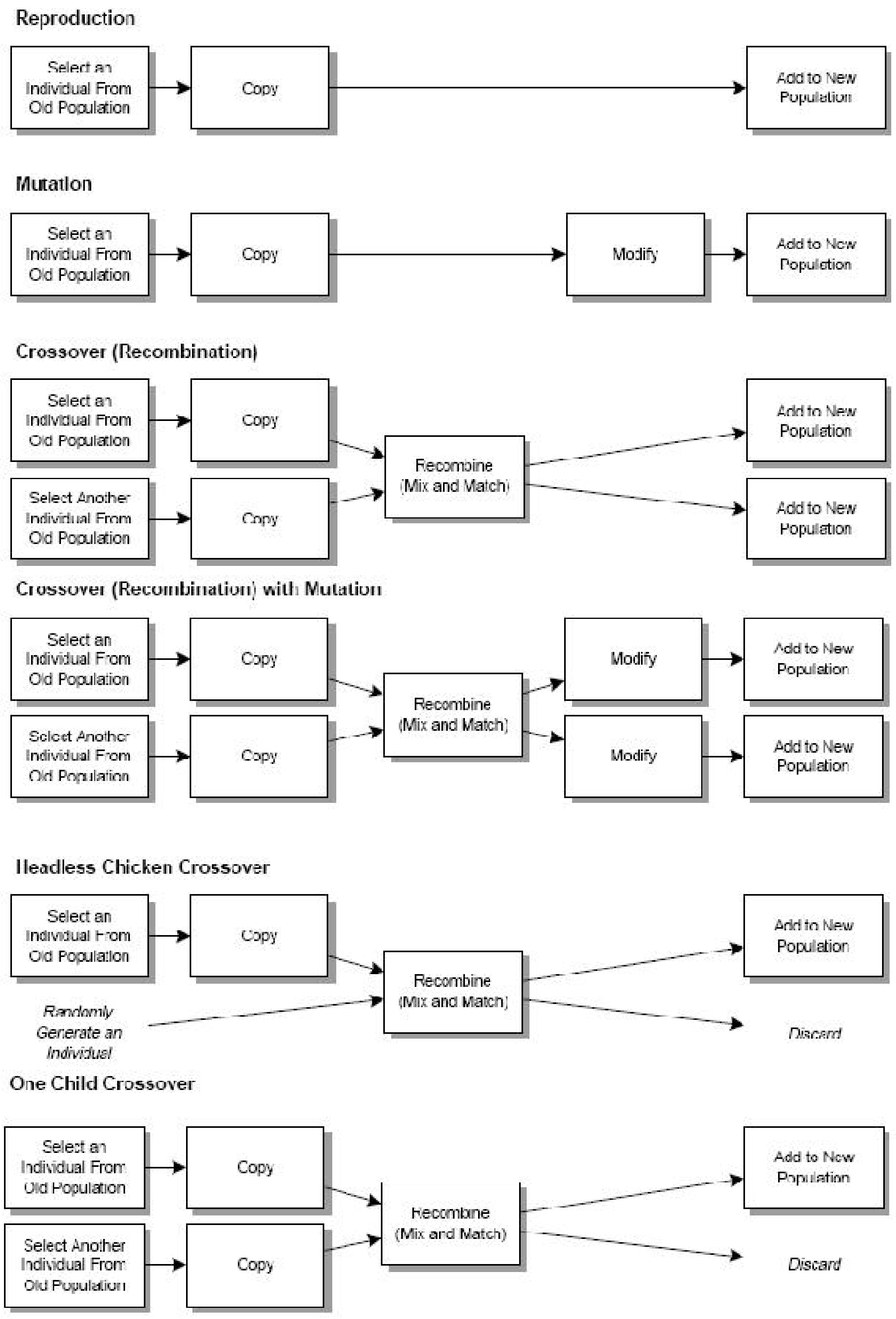

Reproduction
Select an Individual From Old Population
Copy
Add to New Population
Mutation
Select an Individual From Old Population
Copy
Modify
Add to New Population
Crossover (Recombination)
Select an Individual From Old Population
Copy
Select Another Individual From Old Population
Copy
Recombine (Mix and Match)
Add to New Population
Add to New Population
Crossover (Recombination) with Mutation
Select an Individual From Old Population
Copy
Select Another Individual From Old Population
Copy
Recombine (Mix and Match)
Modify
Modify
Add to New Population
Add to New Population
Headless Chicken Crossover
Select an Individual From Old Population
Copy
Randomly Generate an Individual
Recombine (Mix and Match)
Add to New Population
Discard
One Child Crossover
Select an Individual From Old Population
Copy
Select Another Individual From Old Population
Copy
Recombine (Mix and Match)
Add to New Population
Discard

Figure 24 **Six breeding methods in Evolutionary Computation**

The Linkage problem
The "linkage problem" is where we want to have functionally related loci to be more likely to stay together on the string under crossover, but it is not clear how this is to be done without knowing ahead of time which loci are important in useful schemas.
Defining length
The defining length of a schemata is based on the distance between the first and last bits in the scheme with value either 0 or 1 (i.e. not a wild card * symbol). Given that each position in a schema can be 0, 1 or *, then scanning from left to right we can note the following. If I_x is the index of the position of the rightmost occurrence of either a 0 or 1 and I_y is the index of the leftmost occurrence of either a 0 or 1, then the defining length is merely $I_x – I_y$. Thus, the defining length of: **1*1**0**011* is 13 – 3 = 10.

4.4.4.1 Inversion
Along with selection, mutation and crossover, ***inversion*** is often considered to be a basic genetic operator. It is a reordering operator inspired by a similar operator in real genetics. In real genetics, unlike that in simple GAs, the function of a gene is often independent of its position in the chromosome, so inverting part of the chromosome will retain much or all of the "semantics" of the original chromosome. Inversion can change the linkage of bits on the chromosome such that bits with greater nonlinear interactions can potentially be moved closer together.
Typically, inversion is implemented by reversing a random segment of the chromosome. However, before one can start moving bits around on the chromosome to improve linkage, the bits must have a *position independent decoding*. A common error that some researchers make when first implementing inversion is to reverse bit segments of a directly encoded chromosome. But just reversing some random segment of bits is nothing more than large-scale mutation if the mapping from bits to parameters is position dependent.
A *position independent encoding* requires that each bit be tagged in some way. For example, consider the following encoding composed of pairs where the first number is a bit tag that indexes the bit and the second represents the bit value.

[(8 0) (2 1) (9 1) (1 1) (7 0) (5 1) (4 0) (3 1) (6 0)]

Moving around the tag-bit pairs can now change the linkage, but the string remains the same when decoded: 111010001. One must now also consider how recombination is to be implemented.
Inversion works by choosing two points in the string and reversing the order of the bits between them. For example, if the above chromosome was taken and bits 4 and 9 were reversed we would obtain,

[(8 0) (2 1) (9 1) **(6 0)** (7 0) (5 1) (4 0) (3 1) **(1 1)**]

This does not change the fitness of the chromosome, since to calculate the fitness the string is ordered by the indices. However, it does change the linkages and can lead to orderings in which beneficial schemas are more likely to survive, at least, under single-point crossovers.
Inversion does not come without its problems and improvements, if any, may not be justified by the requirements for additional space (to store indices for every bit) and additional computation time (for example, to reorder one parent before crossover) that inversion requires. The problem that comes with single-point inversion is as follows. Suppose, for example that
[(8 0) (2 1) (9 1) (1 1) (7 0) (5 1) (4 0) (3 1) (6 0)]

crosses with
[(7 0) (3 1) (2 1) (9 1) (6 0) (1 1) (5 1) (4 0) (8 0)]

after the third bit, the **offspring** are:
[(8 0) (2 1) (9 1) (9 1) (6 0) (1 1) (5 1) (4 0) (8 0)]

and
[(7 0) (3 1) (2 1) (1 1) (7 0) (5 1) (4 0) (3 1) (6 0)]

The first offspring has two copies each of bits 8 and 9 and no copies of bits 3 and 7. The second offspring has two copies of bits 3 and 7 and no copies of bits 8 and 9.
To ensure that crossover produces offspring with a full set of loci, Holland proposed two possible solutions:

1 Permit crossover only between chromosomes with the same permutation of the loci. This would work but would severely limit the manner in which crossover can be implemented.
2 Employ a "master/slave" approach where one parent is chosen as master and temporarily reorder the other parent to have the same ordering as the master. Use this ordering to reduce offspring, returning the second parent to its original ordering once crossover has been performed.

Both methods have been used in experiments on inversion.

4.4.5 Elitism

This simply consists in storing away the parameters defining the fittest member of the current population, and later copying it intact in the offspring population. Every time a new population is generated, there exists a probability that we might lose the chromosome with the best evaluation. Thus elitism represents a safeguard against the possibility that crossover and/or mutation destroy the current best solution, which would have a high probability of unnecessarily slowing down the optimization process. The adverse effects could be that allows a population to be dominated by a super individual and hence lead to premature convergence.

4.5 Some basic theorems and terms

In order to read the literature on GAs some of the theorems, hypothesis and terms that are likely to be quoted are briefly described in this section.

4.5.1 Schemata and the Schema theorem [39]

Holland's schema theorem [4] was the first rigorous explanation of how GAs function. A ***schema*** is a pattern of gene values that may be represented (in a binary coding) by a string of characters in the following alphabet: {0, 1, *}.

A particular chromosome is said to contain a particular schema if it matches that schemata with the wild card character "*", matching any number 1 or 0. For example, the chromosome "11011" contains, among others, the schemata "11**1," "**011," "****1," "**0*1" and "**01*."

The *order* of a schema is the number of non-* symbols it contains (3, 3, 1, 2, 2) respectively in the above example).

The ***defining length*** of a schema is the distance *between* the outermost non-* symbols (4, 2, 0, 2, 1) respectively in this example.

A chromosome of length *n* bits, will have 3^n different schemas (due to 0, 1 and *).

The **schema theorem** states that *the schemata with above average objective function values will receive exponentially increasing representation in successive populations, especially those schemata that have very few defined bits located close to one another*.
The **schema theorem** explains the power of the GA in terms of how schemata are processed. Individuals in the population are given opportunities to reproduce, often referred to as *reproductive trials,* and produce offspring. The number of such opportunities an individual receives is in proportion to its fitness. Thus, the better individuals contribute more of their genes to the next generation. It is assumed that an individual's high fitness is due to the fact that it contains good schemata. By passing on more of these good schemata to the next generation, increases the likelihood of finding even better solutions.
Holland showed that the optimum way to explore the search space is to allocate reproductive trials to individuals in proportion to their fitness relative to the rest of the population. In this way, good schemata receive an exponentially increasing number of trials in successive generations.
A more succinct definition of this theorem is that:
Short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

4.5.2 Implicit Parallelism Theorem

Holland also showed that, since each individual contains a great many different schemata, *the number of schemata which are effectively being processed simultaneously in each generation is of the order N^3, were N is the population size*. This property is known as **implicit parallelism**, is the cornerstone of the genetic algorithm's power and is one of the explanations for the good performance of GAs.
Implicit parallelism results from the simultaneous evaluation of the numerous building blocks that comprise the string. Each string may contain millions of these building blocks and the genetic algorithm assesses them simultaneously each time it calculates the string's fitness. In effect, the algorithm selects schemata inside the string that exhibit high fitness and passes these building blocks on to the next generation.
The implicit parallelism theorem also shows that *for allelic GAs, more schemata are processed if the alphabet is small.* Historically, this was the motivation for using binary coding in the first GAs.

4.5.3 Building Block Hypothesis

The power of the GA lies in it being able to find good *building blocks*.
A GA seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, consisting of bits that work well together and which tend to lead to improved performance when incorporated into an individual.
Building block hypothesis is just an article of faith as
S1 = 111*********
S2 = **********11
S3 = S2 + S1 = 111*******11 which is no longer a short, low order schemata.
Building block hypothesis recommendations
A successful coding scheme is one that encourages the formation of building blocks by ensuring that:
1. related genes are close together on the chromosome, while
2. there is little interaction between genes.

Interaction or epistasis between genes means that the contribution of a gene to the fitness depends on the value of other genes in the chromosome. For example, picking an orange from the top of a tree requires observing where the orange is located and reaching up to pick it. The possession of

sight alone or of long arms alone is of little use. Therefore, the genes for sight can only increase the "fitness" of an orange picker if it also has genes for long arms.
In fact, there is always some interaction between genes in multimodal (more than one maxima) fitness functions.
If the above two recommendations are observed, then a GA will be as effective as predicted by the schema theorem. However, these rules are difficult to follow. Genes may be related in ways that do not allow all closely related ones to be placed adjacent to each other in a one-dimensional string because of their strong interaction. In many cases the exact nature of the relationship between the genes may not be known to the programmer, so even if there are only simple relationships, it may still be impossible to arrange the coding to reflect this.

4.5.4 Epistasis

The Penguin dictionary of biology [37] has defined the term epistasis as:
"Interaction between non-allelic genetic elements or their products, sometimes restricted to cases in which one element suppresses expression of another *(epistatic dominance).* Analogous to genetic dominance. Segregation of epistatic genes in a cross can modify expected phenotypic ratios among offspring for characters they affect."
In the sense that is used in GAs it generally refers to any kind of strong interaction among genes, not just masking effects.

- Epistasis means that the influence of a gene on the fitness of an individual depends on what gene values are present elsewhere.
- Epistasis is used in the sense of having a masking or switching effect.
- A gene is said to be epistatic when its presence suppresses the effect of a gene at another locus.
- Epistatic genes are sometimes referred to as inhibiting genes because of their effect on other genes which are described as hypostatic (Suppression of expression of a ***(hypostatic)*** gene by another non-allelic gene).

Beasley et al [27] offer their own definition of epistasis in a GA context as,

"Epistasis is the interaction between different genes in a chromosome. It is the extent to which the "expression" (i.e. contribution to fitness) of one gene depends on the values of the other genes…"
They go on to define three levels of gene interaction that depend upon the change in chromosome fitness.

4.5.5 Deception

One of the fundamental principles of GAs is that chromosomes that include schemata, which are contained in the global optimum, will increase in frequency (this is especially true of short, low-order schemata, known as building blocks). Eventually, via the process of crossover, these optimal schemata will come together and the globally optimum chromosome will be constructed. But if schemata which are *not* contained in the global optimum increase in frequency *more* rapidly than those which *are*, the GA will be mislead, away from the global optimum, instead of towards it. This is known as *deception*.
Deception is a special case of epistasis and is directly related to its detrimental effects.

4.6 Implementation Guidelines

Both the schema and implicit parallel theorem imply (indirectly) the following implementation guidelines.

- GA's work best if the optimal solutions are members of schemata that has few defined bits, is packed tightly together and the involved schema has a low standard deviation. That is, since GA's search primarily by combining pieces of good chromosomes, coding should be devised so that similar chromosomes have similar objective function values. In this way, the GA is not deceived in creating chromosomes that are much worse than the sum of their parts.

- For most practical applications, a population size proportional to the chromosome length should suffice [39], although there is never any harm in using a larger population. The implicit parallelism theorem implies that doubling the population may more than halve the run time; thus, if the objective function calculation is fast enough for the use of a large population to be feasible, a large population should be used.

Because of the incomplete nature of GA theory, much knowledge about the successful implementation of GA's comes from experience and experiment.

- For binary-coded GA's, the crossover and mutation rates most often quoted in the literature are $0.6 \le p_{cross} \le 0.9$, and $0.001 \le p_{mut} \le 0.01$. Generally, p_{mut} corresponds to at most the mutation of one or two alleles per chromosome and at least a few chromosomes per generation.
  Real-coded GA's often use higher mutation rates.

- Problems with real design parameters are often (but not always) better handled by real-coded GA's. In general, codings for physical problems work best if they resemble the parameter they model.

- The inclusion of a local optimizer to either:
  1. seed the initial population;
  2. optimize the intermediate populations; or
  3. touch up the final result, can speed the optimization process by removing the local refinement burden from the GA.

- Binary tournament selection generally works faster than roulette-wheel selection, and it avoids convergence troubles [25].

- When the GA stagnates, raising the mutation rate often yields useful information; if the objective function value of the best design changes significantly, the GA may have been temporarily stuck in a local optimum and continued optimization is likely to be useful. If not, restarting the GA with a new initial population will yield a better chance for improvement than attempting to push further.

- Many difficult problems will succumb to GA optimization after the inclusion of an advanced operator, such as elitism.

4.7 References

1. *Genetic Programming*, John R. Koza, MIT Press, 1992.
2. *Primer on Molecular Genetics*, DOE Human Genome Program, Human Genome Management Information System, June, 1992.
3. *Molecular Structure of Nucleic Acids*, J.D. Watson, F.H.C. Crick, Nature, No. 4356, April 25, pp. 737, 1953.
4. *Adaptation in Natural and Artificial Systems*, John H. Holland, University of Michigan Press, Ann Arbor, MI, 1975.
5. *Adaptation in Natural and Artificial Systems*, John H. Holland, MIT Press, 1992.
6. *Genetic Algorithms in Optimization, Search and Machine Learning*, David Goldberg, Addison Wesley, 1989.
7. *Visualization of Genetic Algorithms in a Learning Environment*, Obitko, Marek and Slavík, Pavel, Spring Conference on Computer Graphics, SCCG'99. Bratislava : Comenius University, pp. 101-106, 1999. ISBN 80-223-1357-2.
8. Goldberg, D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, Mass.: Addison-Wesley, 1989.
9. Baker, J. E. 1985. Adaptive Selection Methods for Genetic Algorithms. *Proceedings of an International Conference on Genetic Algorithms and their Application*, Hillsdale, New Jersey, USA: Lawrence Erlbaum Associates Associates, pp. 101-111.
10. Baker, J. E. 1987. Reducing Bias and Inefficiency in the Selection Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 14-21. Lawrence Erlbaum Associates.
11. Gorges-Schleuter, M.: Explicit Parallelism of Genetic Algorithms through Population Structures. *PPSN I. volume 496 of Lecture Notes in Computer Science*, Berlin, Heidelberg, New York: Springer-Verlag, pp. 150-159, 1991.
12. Voigt, H.-M., Santibanez-Koref, I. and Born, J.: Hierarchically Structured Distributed Genetic Algorithm. *Parallel Problem Solving from Nature, 2. Amsterdam*: Elsevier Science Publishers, pp. 145-154, 1992.
13. Mühlenbein, H. and Schlierkamp-Voosen, D.: Analysis of Selection, Mutation and Recombination in Genetic Algorithms. Technical Report 93-24, GMD, 1993.
14. Blickle, T. and Thiele, L.: A Comparison of Selection Schemes used in Genetic Algorithms. TIK Report Nr. 11, December 1995, www.tik.ee.ethz.ch/Publications/TIK-Reports/TIK-Report11abstract.html, 1995.
15. Grefenstette, J. J. and Baker, J. E. 1989. How genetic algorithms work: a critical look at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms*, 20–27. Morgan Kaufmann.
16. Whitley, D. 1989. The GENITOR algorithm and selection pressure: why ranked-based allocation of reproductive trials is best. *Proceedings of the Third International Conference on Genetic Algorithms*, 239–255. Morgan Kaufmann.
17. Schwefel,H.-P. 1977. *Numerische Optimierung vonComputer-Modellen mittels der Evolutionsstrategie ("Numeric Optimization of Computer Models by Means of an Evolution Strategy"), Interdisciplinary System Research*, Volume 26. Bassel: Birkhauser.
18. Rechenberg, I. 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen evolution ("Evolution strategy: the optimization of technical systems according to the principles of biological evolution")*. Stuttgart: Frommann–Holzboog Verlag.
19. Fogel, L. J., Owens, A. J. and Walsh, M. J.: Artificial Intelligence through Simulated Evolution. New York: John Wiley, 1966. [FF93]
20. Angeline, P. J. and Pollack, J. B. 1993. Competitive environments evolve better solutions for complex tasks. In Forrest, S., ed., *Proceedings of the 5th InternationalConference on Genetic Algorithms, ICGA-93*, 264–270. University of Illinois at Urbana-Champaign: Morgan Kaufmann.

21. Cohoon, J. P., Hegde, S. U., Martin, W. N. and Richards, D. S. 1987. Punctuated equilibria: a parallel genetic algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, 148–154. Erlbaum.

22. Wright, S. 1964. Stochastic processes in evolution. In Gurland, J., ed., *Stochastic Models in Medicine and Biology*, 199–241. University of Wisconsin Press.

23. Bäck, T. and Hoffmeister, F. 1991. Extended Selection Mechanisms in Genetic Algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 92-99.

24. Crow, J. F. and Kimura, M.: An Introduction to Population Genetics Theory. New York: Harper and Row, 1970.

25. Goldberg, D. E. and Deb, K. 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. *Foundations of Genetic Algorithms,* San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 69-93.

26. Beasley, D., Bull D.R. and Martin R.R. 1993. An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing, 15(2) pp.58-59.

27. Beasley, D., Bull D.R. and Martin R.R. 1993. An Overview of Genetic Algorithms: Part 2, Research Topics. University Computing, 15(4) pp. 170-181

28. De Jong, K.: An analysis of the behavior of a class of genetic adaptive systems, Doctoral dissertation, University of Michigan, Dissertation Abstracts International, 36(10), 5140B, University Microfilms No. 76-9381, 1975. [DS78]

29. Caruana, R. A., Eshelmann, L. A. and Schaffer, J. D.: Representation and hidden bias II: Eliminating defining length bias in genetic seach via shuffle crossover. in Eleventh International Joint Conference on Artificial Intelligence, Sridharan, N. S. (Ed.), vol. 1, pp. 750-755, San Mateo, California, USA: Morgan Kaufmann Publishers, 1989. [CF94]

30. Mühlenbein, H. and Schlierkamp-Voosen, D.: Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization. Evolutionary Computation, 1 (1), pp. 25-49, 1993.

31. Mühlenbein, H.: The Breeder Genetic Algorithm - a provable optimal search algorithm and its application. Colloquium on Applications of Genetic Algorithms, IEE 94/067, London, 1994.

32. Bäck, T.: Optimal Mutation Rates in Genetic Search, In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, USA: Morgan Kaufmann Publishers, 1993., pp. 2-8, 1993.

33. Syswerda, G.: Uniform crossover in genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 2-9, 1989.

34. Eshelmann, L. A., Caruana, R. A. and Schaffer, J. D.: Biases in the crossover landscape. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 10-19, 1989.

35. Spears, W.M. and De Jong, K. A.: An Analysis of Multi-Point Crossover. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 301-315, 1991.

36. Davis, L. D., editor. 1991. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

37. M. Abercrombie et al, 1990. *The New Penguin Dictionary of Biology.* Penguin Books.

38. Darrel Whitley. *A Genetic Algorithm Tutorial*. Technical Report CS-93-103, March 10, 1993. Colorado State University. Published as: D. Whitley. A genetic algorithm tutorial. Statistics and Computing (1994) 4: 65-85.

39. D. E. Goldberg, K. Deb, and J. H. Clark, “Genetic algorithms, noise, and the sizing of populations,” *Complex Syst.*, vol. 6, no. 3, pp. 333–362, 1991

Appendix 1

Unsigned Fixed-Point Rationals

An N-bit binary word, when interpreted as an unsigned fixed-point rational, can take on values from a subset *P* of the non-negative rationals given by

$$P = \left\{ p/2^b \,\middle|\, 0 \le p \le 2^N - 1, p \in \quad \right\} \quad \text{where} \quad \text{is the set of integers}$$

Note that *P* contains 2^N elements. We denote such a representation *U*(*a,b*), where *a* = *N* - *b*.
In the *U*(*a,b*) representation, the *n*th bit, counting from right to left and beginning at 0, has a weight of $2^n/2^b = 2^{n-b}$. Note that when *n* = *b* the weight is exactly 1. Similar to normal everyday base-10 decimal notation, the binary point is between this bit and the bit to the right. This is sometimes referred to as the implied binary point. A *U*(*a,b*) representation has *a* integer bits and *b* fractional bits.
The value of a particular N-bit binary number *x* in a *U*(*a,b*) representation is given by the expression

$$x = \left(1/2^b\right) \sum_{n=0}^{N-1} 2^n x_n$$

where x_n represents bit *n* of *x*. The range of a *U*(*a,b*) representation is from 0 to $\left(2^N - 1\right)/2^b = 2^a - 2^{-b}$

For example, the 8-bit unsigned fixed-point rational representation *U*(6,2) has the form

$$b_5 b_4 b_3 b_2 b_1 b_0 b_{-1} b_{-2}$$

where bit b_k has a weight of 2^k. Note that since *b* = 2 the binary point is to the right of the second bit from the right (counting from zero), and thus the number has six integer bits and two fractional bits.
This representation has a range of from 0 to $2^6 - 2^{-2} = 64 - 1/4 = 63\ \ 3/4$.
The unsigned integer representation can be viewed as a special case of the unsigned fixed-point rational representation where *b* = 0. Specifically, an N-bit unsigned integer is identical to a *U*(*N*,0) unsigned fixed-point rational. Thus the range of an N-bit unsigned integer is

$$0 \le U\left(N, 0\right) \le 2^N - 1$$

and it has N integer bits and 0 fractional bits. The unsigned integer representation is sometimes referred to as "natural binary."
Examples:
1. *U*(6,2). This number has 6+2 = 8 bits and the range is from 0 to $2^6 - 1/2^2 = 63.75$. The value 8Ah (1000,1010b) is

$$(1/2^2)(2^1 + 2^3 + 2^7) = 34.5$$

2. *U*(-2,18). This number has -2 + 18 = 16 bits and the range is from 0 to $2^{-2} - 1/2^{18} =$ 0.2499961853027. The value 04BCh (0000,0100,1011,1100b) is

$(1/2^{18})(2^2 + 2^3 + 2^4 + 2^5 + 2^7 + 2^{10}) = 1212/2^{18} = 0{:}004623413085938$

3. *U*(16,0). This number has 16 + 0 = 16 bits and the range is from 0 to $2^{16} - 1 = 65{,}535$. The value 04BCh (0000,0100,1011,1100b) is

$(1/2^0)(2^2 + 2^3 + 2^4 + 2^5 + 2^7 + 2^{10}) = 1212/2^0 = 1212.$