AIES Notes: Genetic Algorithms [Lecture 5]

The family of local search algorithms includes methods inspired by statistical physics (simulated annealing) and evolutionary biology (**genetic algorithms**).

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

In many problems, the path to the goal is irrelevant. For example, integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management.

If the path to the goal does not matter, we might consider a different class of algorithms, ones that do not worry about paths at all. **Local search** algorithms operate using a single **current node** (rather than multiple paths) and generally move only to neighbors of that node. Typically, the paths followed by the search are not retained. Although local search algorithms are not systematic, they have two key advantages:

(1) they use very little memory—usually a constant amount; and
(2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

In addition to finding goals, local search algorithms are useful for solving **pure optimization problems**, in which the **aim is to find the best state according to an objective function**.

To understand local search, it is useful to consider the state-space landscape (as in Figure 4.1). A landscape has both "**location**" (defined by the state) and "**elevation**" (defined by the value of the heuristic cost function or objective function). If elevation corresponds to cost, then the aim is to find the lowest valley—a **global minimum**; if elevation corresponds to an **objective function**, then the aim is to find the **highest peak—a global maximum**. (You can convert from one to the other just by inserting a minus sign.) Local search algorithms explore this landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a **global minimum/maximum**.

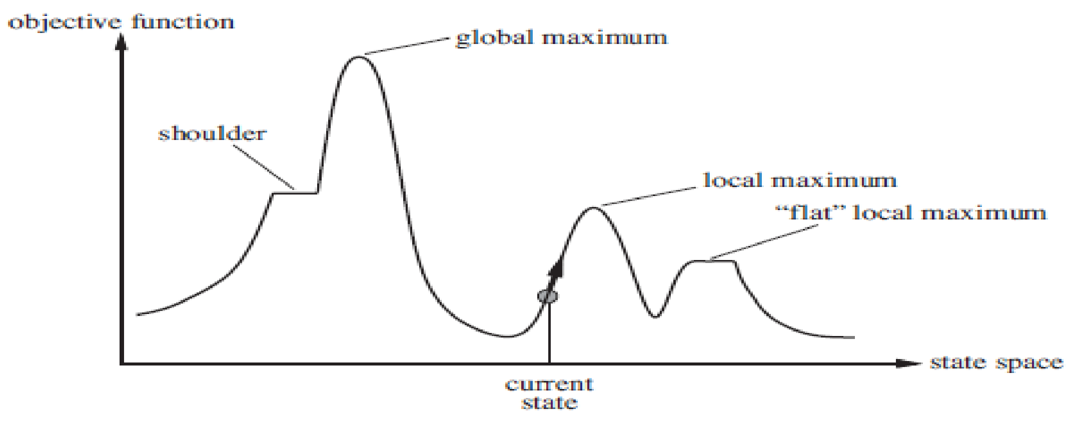


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

There are many local search algorithms namely, **Hill-climbing Search, Simulated Annealing, Local Beam Search,** and **Genetic Algorithms.**

<u>Genetic Algorithms</u>

A genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state. The analogy to natural selection is the same as in stochastic beam search, except that now we are dealing with sexual rather than asexual reproduction.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

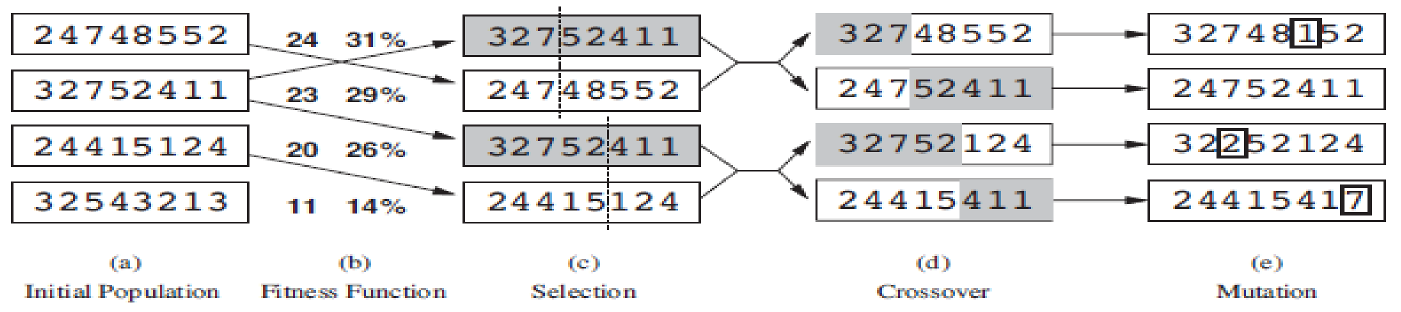


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic Algorithms Flow-Chart

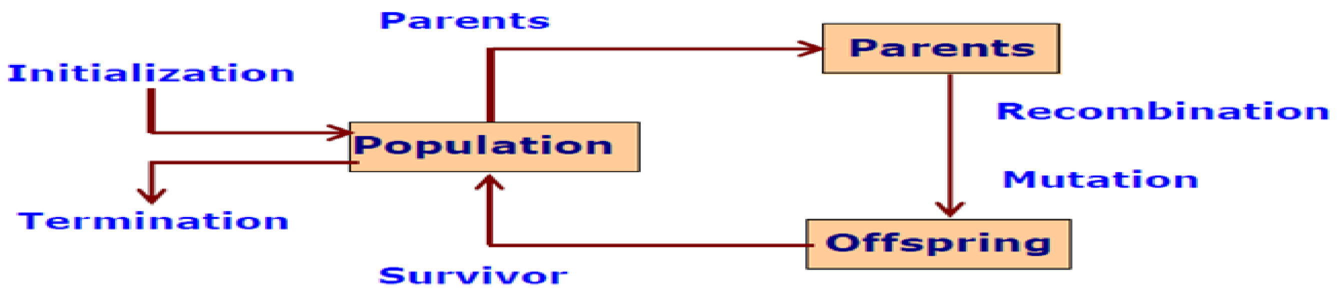


Fig. General Scheme of Evolutionary process

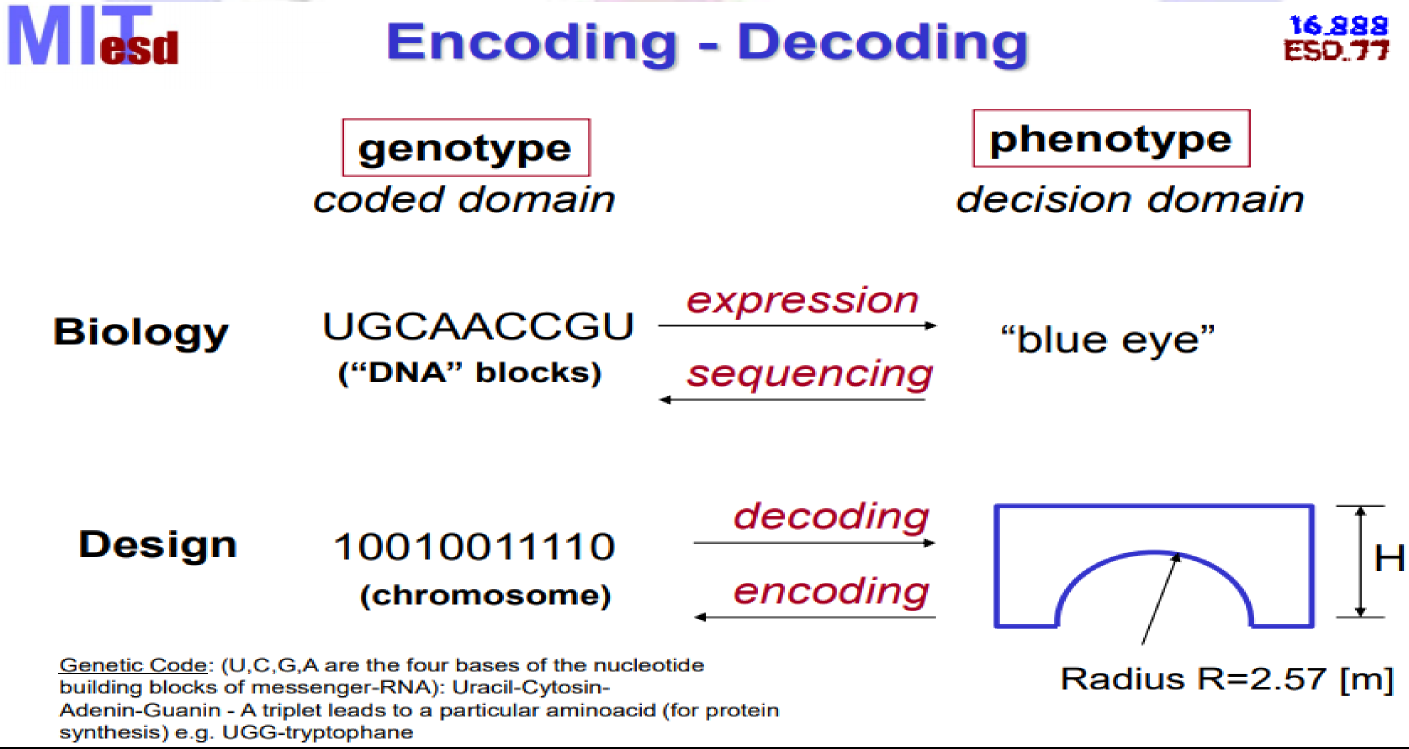
Natural Selection

Charles Darwin (1809-1882) Controversial and very influential book (1859) On the origin of species by means of natural selection, or the preservation of favored races in the struggle for life

Observations:

- Species are continually developing (**evolution**)
- Homo sapiens and apes have common ancestors
- Variations between species are enormous
- Huge potential for production of offspring, but only a small/moderate percentage survive

Encoding-Decoding



Genetic Code: (U,C,G,A are the four bases of the nucleotide building blocks of messenger-RNA): Uracil-Cytosin-Adenin-Guanin - A triplet leads to a particular aminoacid (for protein synthesis) e.g. UGG-tryptophane

Binary Encoding

Binary encoding is the most common to represent information contained. In genetic algorithms, it was first used because of its relative simplicity.
- In binary encoding, every chromosome is a string of bits : 0 or 1, like
Chromosome 1: 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1
Chromosome 2: 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1
- Binary encoding gives many possible chromosomes even with a small number of alleles ie possible settings for a trait (features).
- This encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Operators of GA

Genetic operators used in genetic algorithms maintain genetic diversity. Genetic diversity or variation is a necessary of the process of evolution. Followings are the genetic operators analogous to those which occur in the natural world:

1. Reproduction (or Selection)
2. Crossover (Recombination)
3. Mutation

Fitness Function

• Choosing the right fitness function is very important, but also quite difficult
• GAs do not have explicit "constraints"
• Constraints can be handled in different ways:
– via the fitness function – penalty for violation
– via the selection operator ("reject constraint violators")
– implicitly via representation/coding e.g. only allow representations of the TSP that correspond to a valid tour
– Implement a repair capability for infeasible individuals
Choosing the right fitness function: an important genetic algorithm design Issue

Selection

Many reproduction operators exist and they all essentially do same thing. They pick from current population the strings of above average and insert their multiple copies in the mating pool in a probabilistic manner.

The most used methods of selecting chromosomes for parents to crossover are:

1. Roulette wheel selection,
2. Boltzmann selection,
3. Tournament selection,
4. Rank selection
5. Steady state selection.

Roulette wheel selection

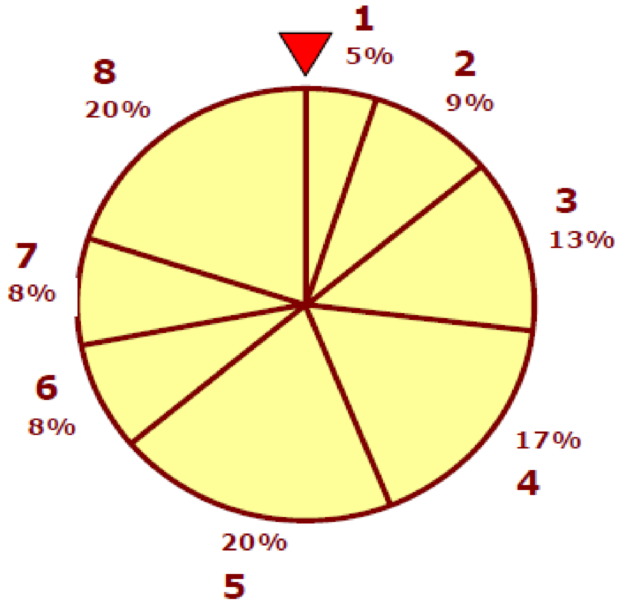


Fig. Roulette-wheel Shows 8 individual with fitness

The Roulette-wheel simulates **8** individuals with fitness values **F_i**, marked at its circumference; e.g.,

- the **5th** individual has a higher fitness than others, so the wheel would choose the **5th** individual more than other individuals .
- the fitness of the individuals is calculated as the wheel is spun **n = 8** times, each time selecting an instance, of the string, chosen by the wheel pointer.

Probability of i^{th} string is $p_i = F_i \,/\, (\sum_{j=1}^{n} F_j)$, where

n = no of individuals, called population size; **pi = probability** of **i^{th}** string being selected; **F_i = fitness** for **i^{th}** string in the population. Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the i^{th} string.

Average fitness = $\bar{F}$ F_j / n ; **Expected count = (n =8) x pi**

Cumulative Probability₅ $= \sum_{i=1}^{N=5} p_i$

Crossover

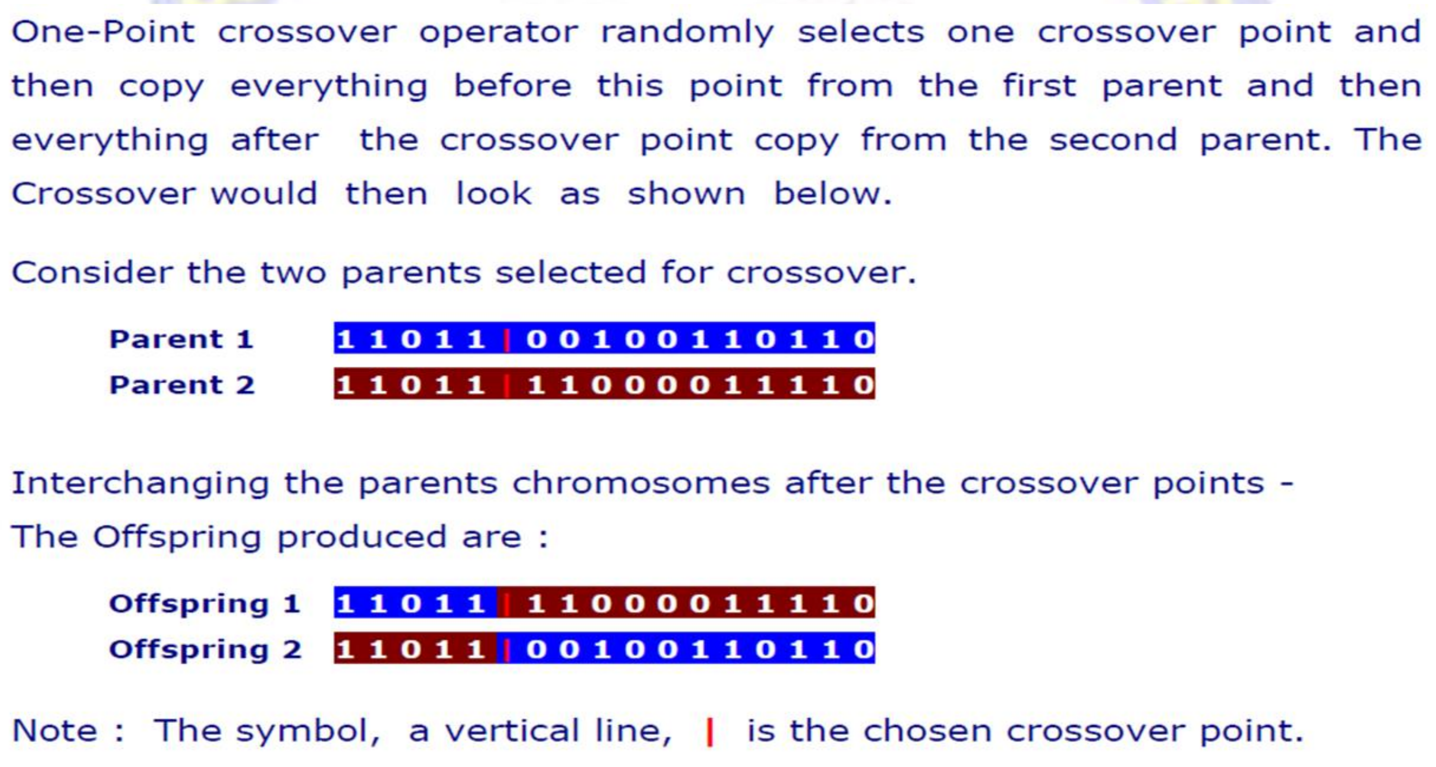
Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring.

The Crossover operators are of many types. - one simple way is,

1. One-Point crossover
2. Two Point
3. Uniform
4. Arithmetic, and
5. Heuristic crossovers.

The operators are selected based on the way chromosomes are encoded.

One-Point Crossover



One-Point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

Consider the two parents selected for crossover.

Parent 1	1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0
Parent 2	1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0

Interchanging the parents chromosomes after the crossover points -

The Offspring produced are :

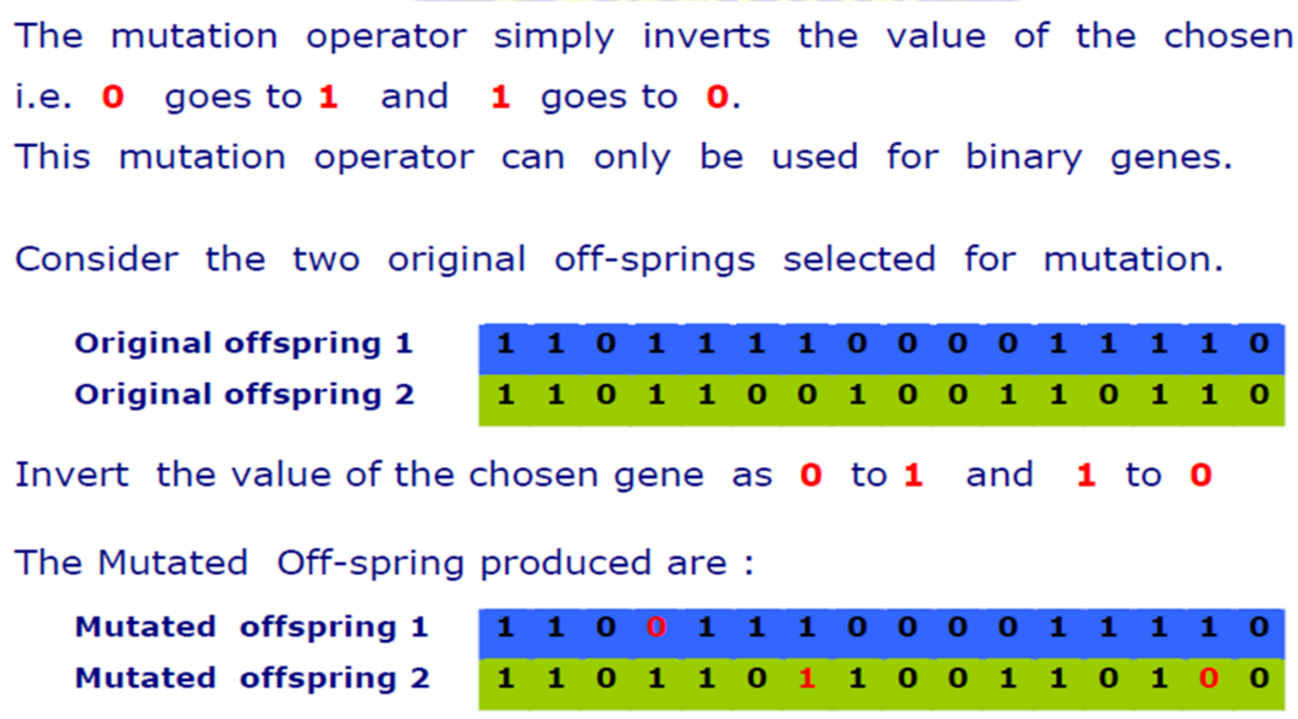
Offspring 1	1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0
Offspring 2	1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0

Note : The symbol, a vertical line, | is the chosen crossover point.

Mutation

After a crossover is performed, mutation takes place. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say 0.01 a good first choice. Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible.

Flip-Bit Mutation



The mutation operator simply inverts the value of the chosen gene. i.e. 0 goes to 1 and 1 goes to 0.

This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

Original offspring 1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0
Original offspring 2	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0

Invert the value of the chosen gene as 0 to 1 and 1 to 0

The Mutated Off-spring produced are :

Mutated offspring 1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0
Mutated offspring 2	1	1	0	1	1	0	1	1	0	0	1	1	0	1	0	0