Artificial Intelligence

Dr. Qaiser Abbas
Department of Computer Science & IT
University of Sargodha

Genetic algorithms (Ref. B. 1)

- The genetic algorithm isn't really a single algorithm, but a collection of algorithms and techniques that can be used to solve a variety of problems in a number of different problem domains.

- Lets first discuss shortly its evolution.

Evolutionary Strategies

- In early evolutionary strategy, the population size was restricted to two members, the parent and child.

- The child member was modified in a random way (a form of mutation), and whichever member was more fit (parent or child) was then allowed to propagate to the next generation as the parent.

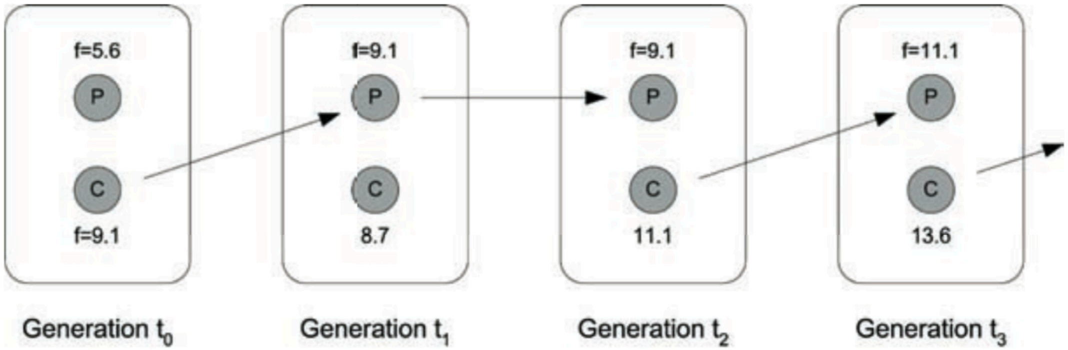


FIGURE 7.1: **Demonstrating the simple two member evolutionary strategy.**

Evolutionary Strategies

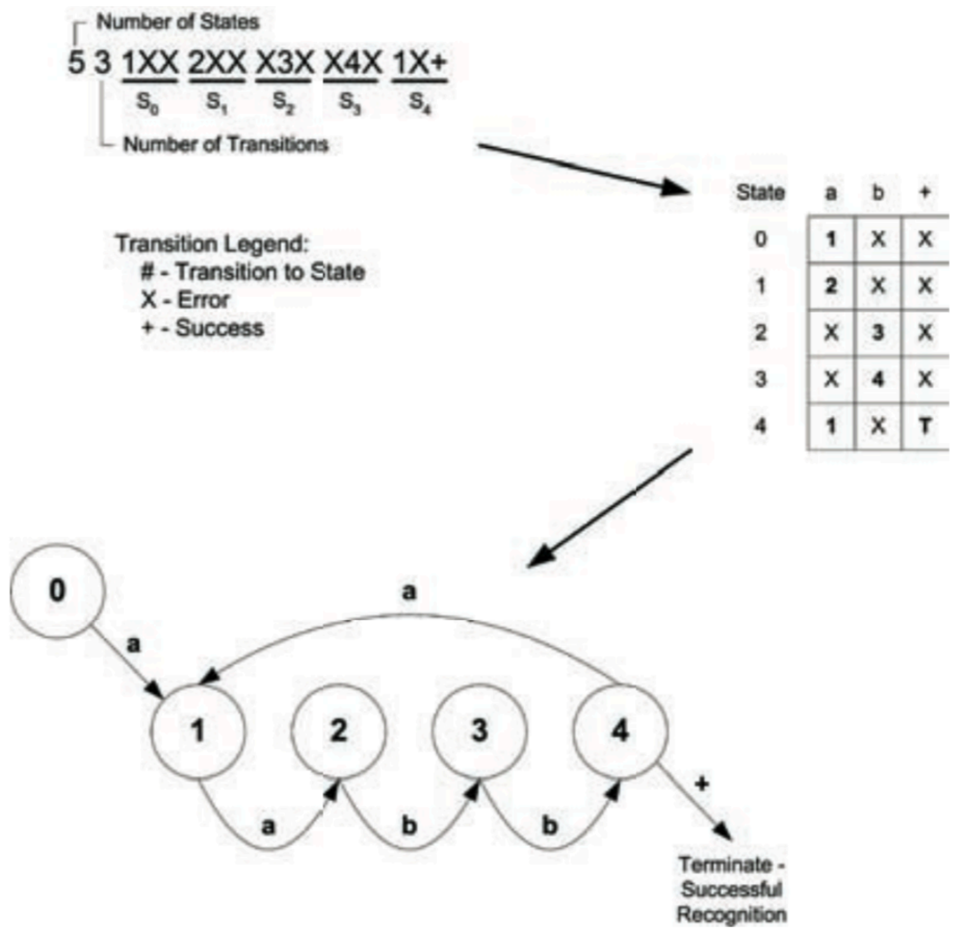


FIGURE 7.2: **Evolving finite state machines for a simple parsing task.**

Genetic Algorithms

- John Holland introduced the idea of genetic algorithms in the 1960s as a population-based algorithm with greater biological plausibility (reasonable) than previous approaches.
- Evolutionary strategies used mutation as a way to search the solution space, Holland's genetic algorithm extended this with additional operators straight from biology.
- In addition to mutation, Holland also used crossover and inversion to navigate the solution space (see Figure 7.3).
- Potential solutions (or chromosomes) are represented as strings of bits instead of real values.

All living organisms consist of cells, where each cell contains a set of chromosomes (strings of DNA). Each chromosome is made up of genes, each of which can encode a trait (behavioral or physical). These chromosomes serve as the basis for genetic algorithms, where a potential solution is defined as a chromosome, and the individual elements of the solution are the genes.

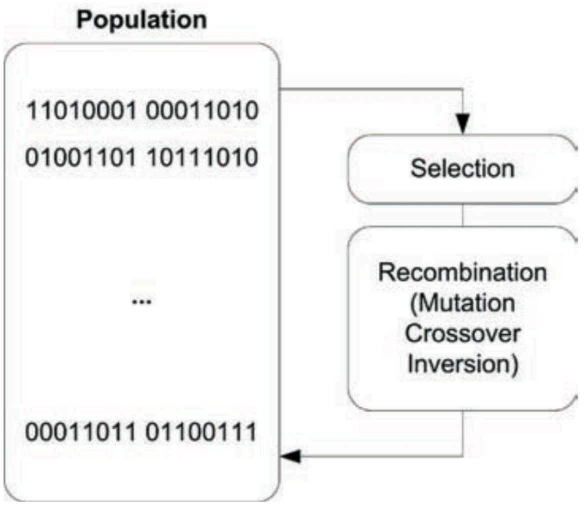


FIGURE 7.3: **Holland's bit-string genetic algorithm.**

Genetic Programming

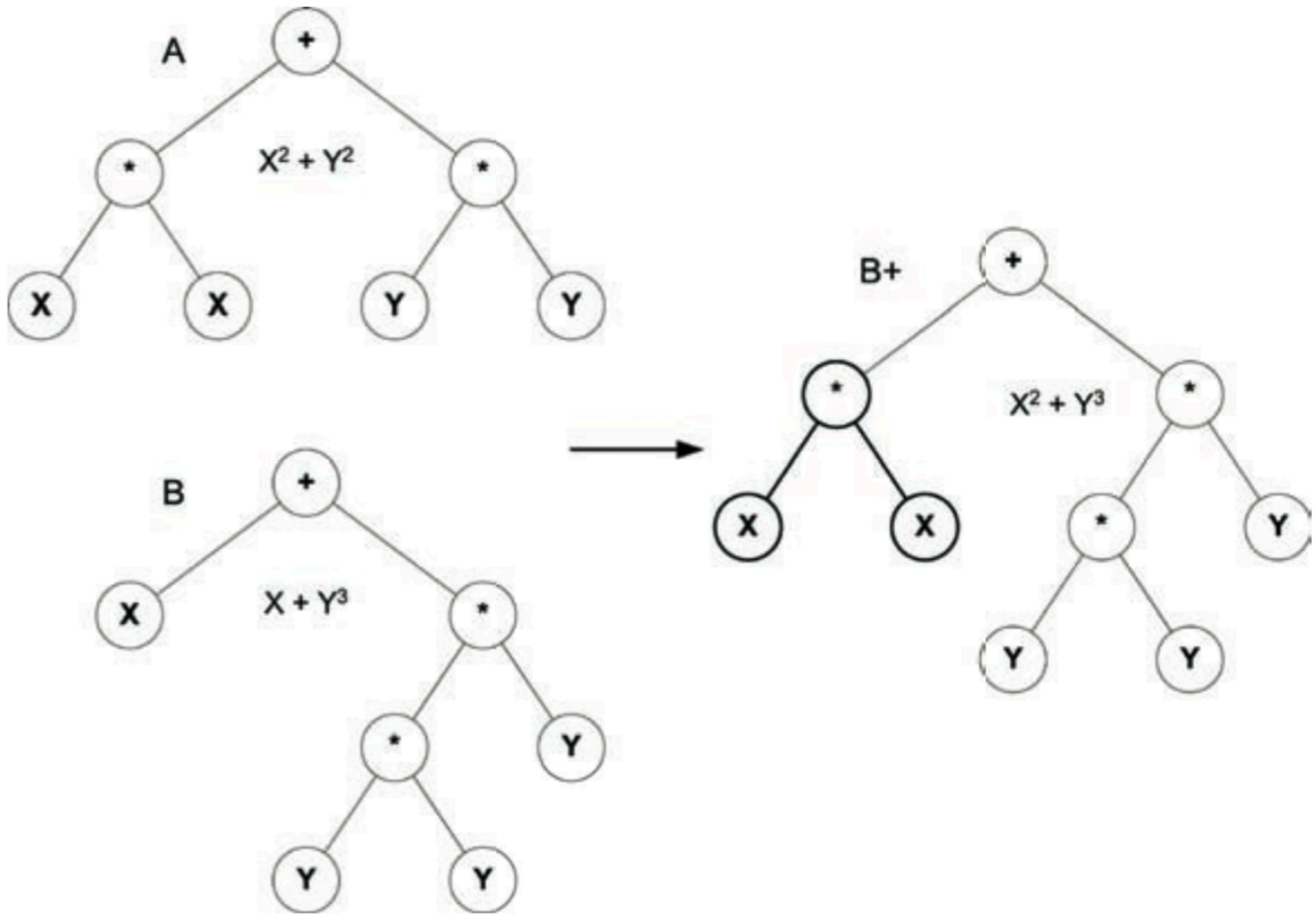


FIGURE 7.4: **Using the crossover operator to create new S-expressions.**

GENETIC ALGORITHMS (GA)

- The GA is called a population-based technique because instead of operating on a single potential solution, it uses a population of potential solutions.
  - The larger the population, the greater the diversity of the members of the population, and the larger the area that is searched.

GENETIC ALGORITHMS (GA)

- One attempt to understand why genetic algorithms work is called the Building-Block Hypothesis (BBH).
- This specifies, for binary GA, that the crossover operation (splitting two chromosomes and then swapping the tails) improves the solution.
- One can think of this as genetic repair, where fit building blocks are combined together to produce higher fitness solutions.
- Additionally, using fitness-proportionate selection (higher fit members are selected more often), less fit members and their corresponding building blocks die out and thus increasing the overall fitness of the population.

GENETIC ALGORITHMS (GA)

- The overall genetic algorithm can be defined by the simple process shown in Figure 7.7.
- First, a pool of random potential solutions is created that should have adequate diversity.
- Next, the fitness of each member is computed.
- Next, members of the population are selected based on some algorithm. The two simplest approaches are roulette wheel selection, and elitist selection (see Figure 7.8).

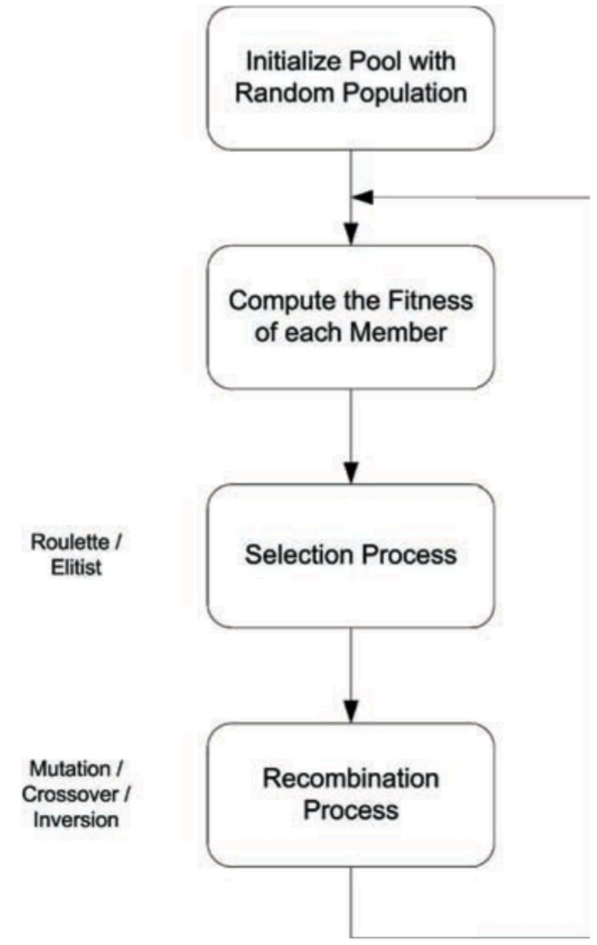


FIGURE 7.7: **Simple flow of the genetic algorithm.**

GENETIC ALGORITHMS (GA)

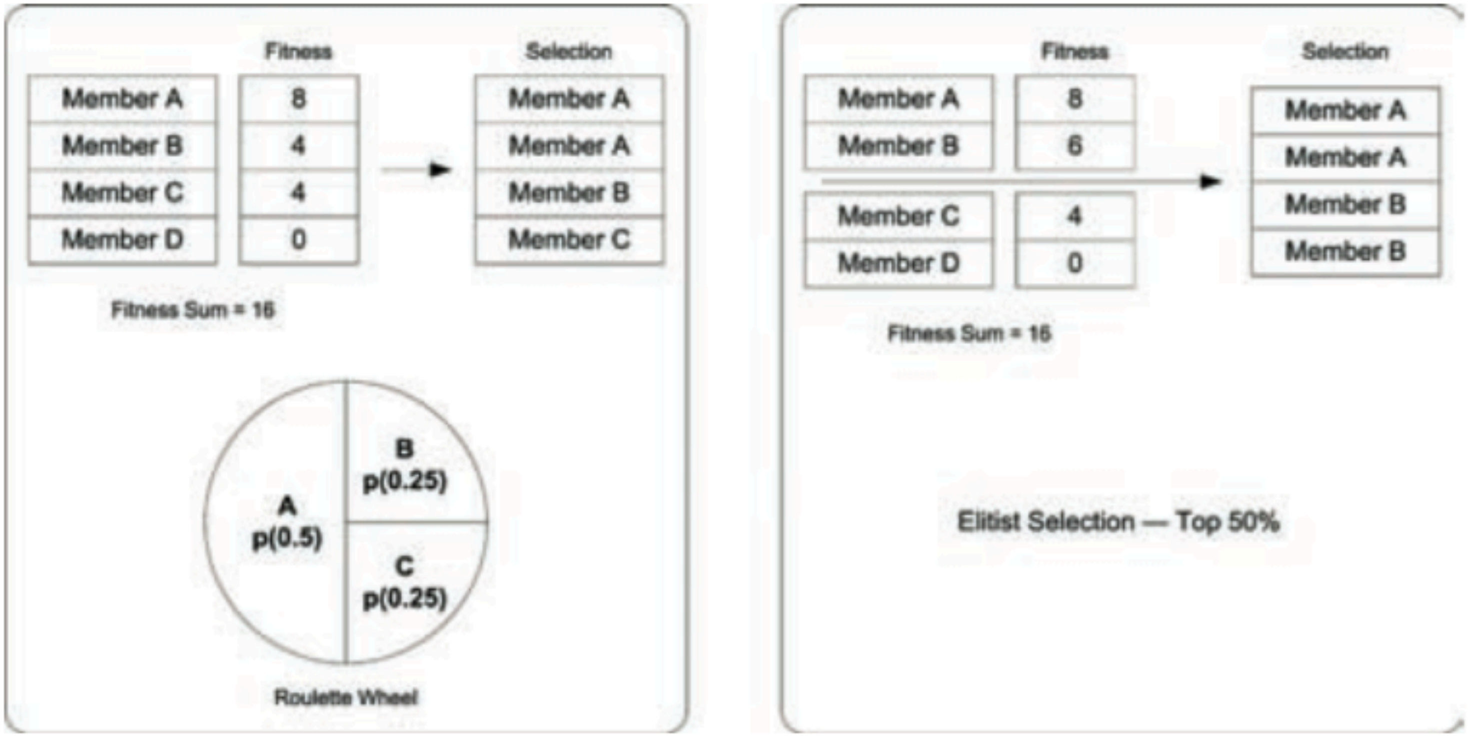


FIGURE 7.8: Two of the simpler GA selection models.

- From the selection process, we have a number of members that have the right to propagate their genetic material to the next population.

GENETIC ALGORITHMS (GA)

- The next step is to recombine these members' material to form the members of the next generation.
- Commonly, parents are selected two at a time from the set of individuals that are permitted to propagate (from the selection process).
- Given two parents, two children are created in the new generation with slight alternations courtesy of the recombination process (with a given probability that the genetic operator can occur). Figures 7.9 and 7.10 illustrate four of the genetic operators.

GENETIC ALGORITHMS (GA)

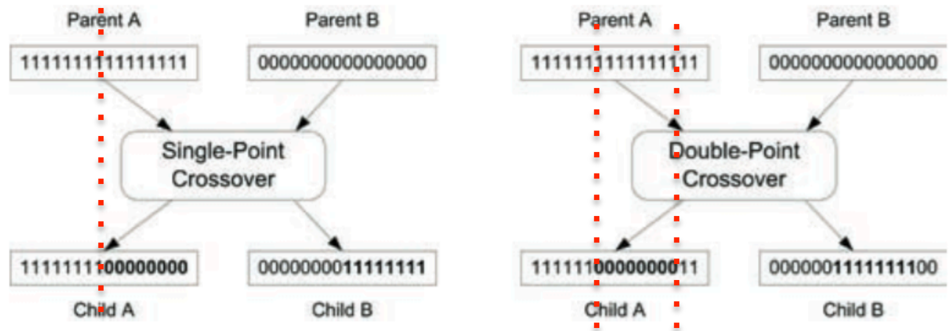


FIGURE 7.9: **Illustrating the crossover operators in genetic recombination.**

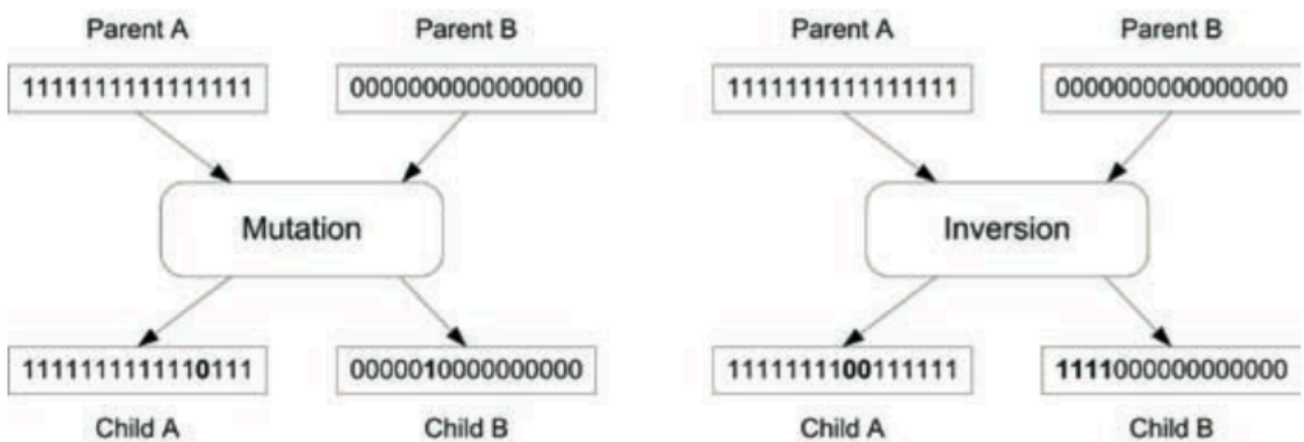


FIGURE 7.10: **Illustrating the mutation and inversion genetic operators.**

GENETIC ALGORITHMS (GA)

- Finally, **how it terminates**? There are a number of ways that we can terminate the process.

  – The most obvious is to end when a solution is found, or one that meets the designer's criteria. But from the algorithm's perspective, we also need to account for the population, and its ability to find a solution.

GENETIC ALGORITHMS (GA)

- Another termination criterion, potentially returning a suboptimal solution, is when the population lacks diversity, and therefore the inability to adequately search the solution space. When the members of the population become similar, there's a loss in the ability to search. To combat this, we terminate the algorithm early by detecting if the average fitness of the population is near the maximum fitness of any member of the population.

The issue of lack of diversity in genetic algorithms results in premature convergence, as the members converge on a local maximum, not having found the global maximum. Early termination is one solution, but others include algorithm restart if this situation is detected.

GENETIC ALGORITHMS (GA)

```
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))
```

GA: Example 1

- let us have a look at a simple genetic algorithm and use it in an everyday application: searching for a maximum of the function , where x can take values between 0 and 31. It is clear right away that the solution is the value x = 31.

- Next, a Simple execution of GA shows the essential element of its operation, which is that the result of the optimization is improving from generation to generation.

GA: Example 1

A series of numbers	Starting population (random)	Value of the variable x	$f(x) = x^2$ f_i	Offspring number fitness (f_i/f_{avg})	Actual offspring number (rounded up)
1	0 1 1 0 1	13	169	0.58	1
2	1 1 0 0 0	24	576	1.97	2
3	0 1 0 0 0	8	64	0.22	0
4	1 0 0 1 1	19	361	1.23	1
Sum			**1170**	**4.00**	**4**
Average: f_{avg}			**293**	**1.00**	**1**
Maximum			**576**	**1.97**	**2**

Table a: Computation of randomly chosen population variables

GA: Example 1

First generation offspring	Randomly chosen cross-over partner	Cross-over point (random)	New population	Value of the variable **x**	$f(x) = x^2$ **f_i/f_{avg}** (rounded up)
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144 (0.32, 0)
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625 (1.42, 1)
0 1 | 0 0 0	4	2	**0** 1 0 1 1	**11**	**121 (0.42, 0)**
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256 (0.58, 1)
Sum					**1146**
Average: f_{avg}					**287**
Maximum					**625**

Table b: The presentation of the cross-over and the first generation offspring characteristics computation

GA: Example 1

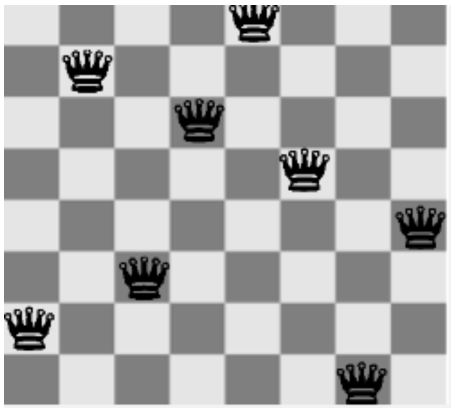
- We have assumed the value 0.001 e.g. 1/1000 bits for the mutation.
- Since there are four subjects in each generation, each of the subjects being five bits (binary places) long, the probability that one of them would mutate is 4*5*0.001=0.02 (2/100).
- In the first step, none of the bits has mutated. We can keep computing in the same way until we reach the value x = 31 in just a few iterations.

A simple example of genetic algorithm

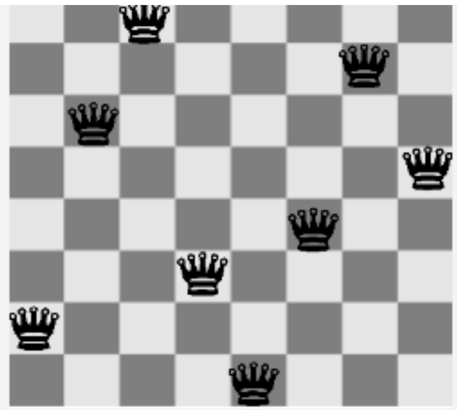
0	1	1	0	0
1	1	0	0	1
1	1	0	1	1
1	0	0	0	0

Random Selection	Crossover Point(Rand)	New Pop	X
3	3	01111	15
4	1	10000	16
1	3	11000	24
2	1	11001	25
2	1	00000	0
1	1	11111	31
4	1	11001	25
3	1	11000	24

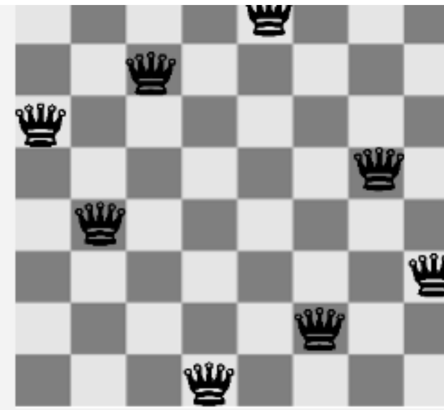
GA: Example 2



[1 6 2 5 7 4 0 3]

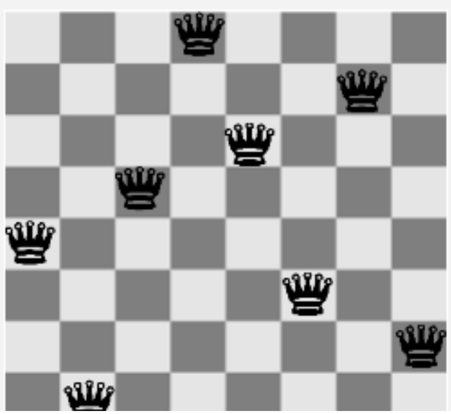


[1 5 7 2 0 3 6 4]

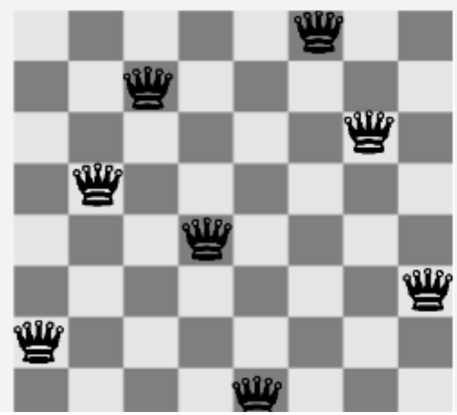


[5 3 6 0 7 1 4 2]

[3 0 4 7 5 2 6 1]



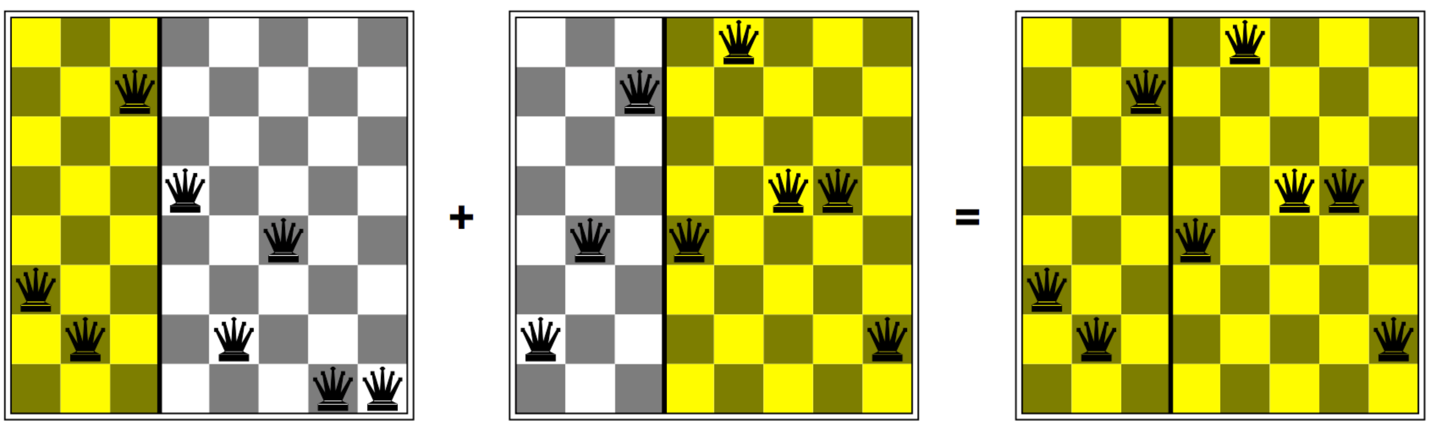
[1 4 6 3 0 7 5 2]



GA: Example 2

- GAs require that the states are encoded as strings.

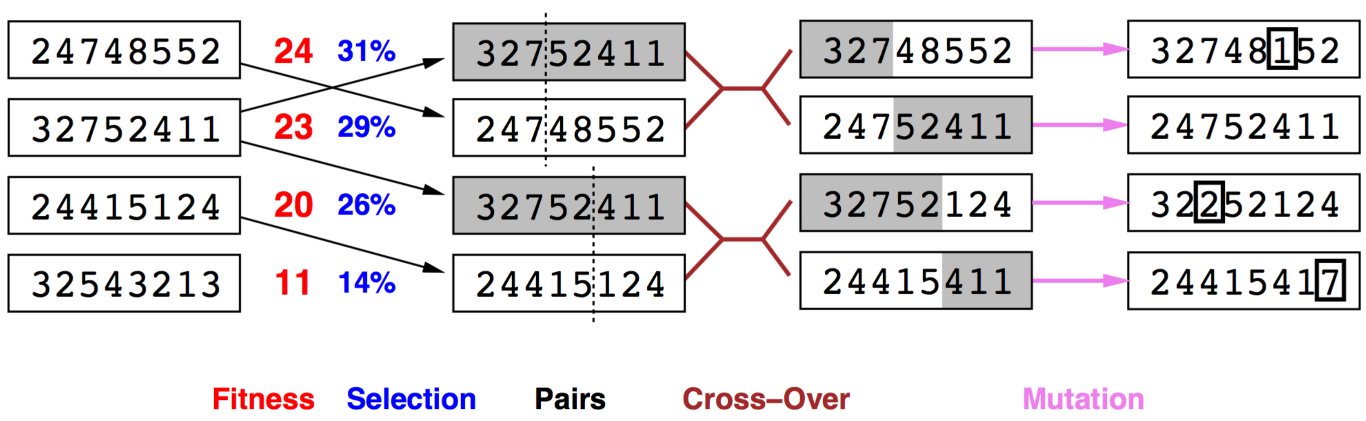
- The crossover helps iff substrings are meaningful components



3 2 7 5 2 4 1 1 + 2 4 7 4 8 5 5 2 = 3 2 7 4 8 5 5 2

GA: Example 2

- Idea:
  - a variant of stochastic local beam search
  - generate successors from pairs of states
  - the states have to be encoded as strings



Additional Reading from TB.Ch.4

- Contingency
- Online search

Lab Project 5

- Implement a basic binary genetic algorithm for a given problem
-
- Visit the following link and study and understand the genetics algorithm. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3
-
- Java source code is given there, configure it, execute it and submit the report accordingly.