Computational Intelligence

Unit # 3

Components of Evolutionary Algorithms

- Representation (definition of individuals)
- Evaluation function (or fitness function)
- Population
- Parent selection mechanism
- Variation operators, recombination and mutation
- Survivor selection mechanism (replacement)

Representation

- The first step in defining an EA is to link the "real world" to the "EA world".
- Objects forming possible solution within the original problem context are referred to as **phenotypes**, while their encoding in the EA are called **genotypes**.
- Within the EC literature many synonyms can be found:
  - **Candidate solution** and **individuals** denote points in the **phenotype space**.
  - **Chromosome** and **individuals** denote points in the **genotype space**.
  - A placeholder is commonly called a variable, a **locus**, or – in a biology-oriented terminology – a **gene**. An object in such a place can be called a value or an **allele**.

Evaluation Function

- The role of the evaluation function is to represent the requirements the population should adopt to.
- Technically, it is a function or procedure that assigns a quality measure to genotypes.
- The evaluation function is commonly called the **fitness function** in EC. This might cause a counterintuitive terminology if the original problem requires minimization, because the term fitness is usually associated with maximization.

Parent Selection Mechanism

- The role of parent selection is to distinguish among individuals based on their quality, and, in particular, to allow the better individuals to become parents of the next generation.
- An individual is a **parent** if it has been selected to undergo variation in order to create offspring.
- In EC, parent selection is typically probabilistic.

Variation Operators

- The role of **variation operators** is to create new individuals from old ones.
- In the corresponding phenotype space this amounts to generating new candidate solutions.
- Variation operators in EC are divided into two types based on their **arity**:
  - Mutation
  - Crossover

Mutation

- Mutation is a unary variation operator.
- It is applied to one genotype and delivers a (slightly) modified mutant, the **child** or **offspring**.
- A mutation operator is always stochastic: its output – the child – depends on the outcomes of a series of random choices.

Crossover

- A binary variation operators is called **recombination** or **crossover**.
- This operator merges information from two parent genotypes into one or two offspring genotypes.
- Like mutation, recombination is a stochastic operator: the choices of what parts of each parent are combined and how this is done, depend on random drawings.
- The principle behind recombination is simple – by mating two individuals with different but desirable features, we can produce an offspring that combines both of those features.

Crossover OR Mutation?

- Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem
- Exploitation: Optimizing within a promising area, i.e. using information
- There is co-operation AND competition between them
- Crossover is explorative, it makes a big jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random small diversions, thereby staying near (in the area of) the parent
- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)

Survival Selection Mechansim

- The role of survival selection (**replacement**) is to distinguish among individuals based on their quality.
- It is similar to parent selection, but it is used I a different stage of the evolutionary cycle.
- In contrast to parent selection, which is typically stochastic, survivor selection is often deterministic.

Initialization and Termination

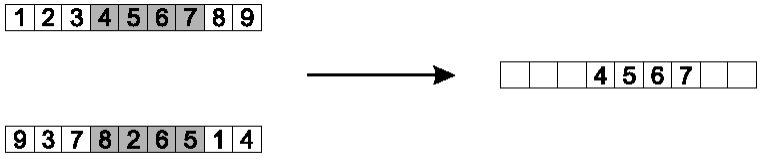
- Initialization is kept simple in most EA applications, the first population is seeded by randomly generated individuals.
- Problem specific heuristics can also be used in this step to create an initial population with higher fitness.
- The following options are used for **termination**:
  - The maximally allowed CPU time elapses.
  - The fitness improvement remains under a threshold value for a given period of time.
  - The population diversity drops under a given threshold.

Order 1 Crossover

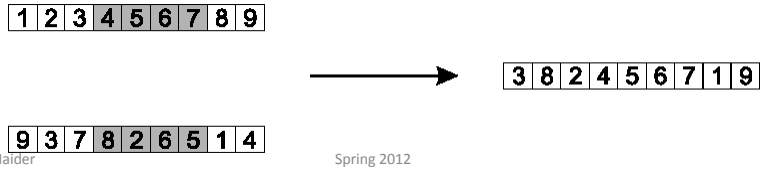
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order** of the second parent
     - and wrapping around at the end
  4. Analogous for the second child, with parent roles reversed

Order 1 Crossover Example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Compute Fitness

Candidate Solutions	Fitness
A C B F H D E G	43
H B G E A C D F	52
A H G C B D F E	49
E G B C D H F A	47
F H A D C B E G	49
C D B A H E G F	56

	A	B	C	D	E	F	G	H
A	0	8	3	1	4	9	3	6
B	8	0	5	10	11	4	3	6
C	3	5	0	8	7	1	5	12
D	1	10	8	0	9	11	6	4
E	4	11	7	9	0	5	17	3
F	9	4	1	11	5	0	4	1
G	3	3	5	6	17	4	0	7
H	6	6	12	4	3	1	7	0

Crossover Example

- We selected the parents in the previous slides

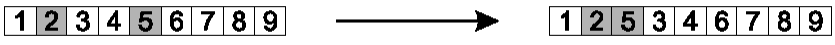
Parent 1	A C B F H D E G	43
Parent 2	F H A D C B E G	49

- Let's do crossover to produce two offspring. Suppose the crossover points are 3 and 5

Offspring 1	D C B F H E G A
Offspring 2	F H A D C E G B

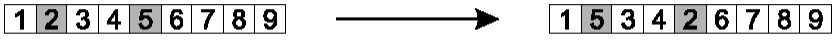
Insert Mutation for Permutations

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information



Swap Mutation for Permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more



Mutation Example

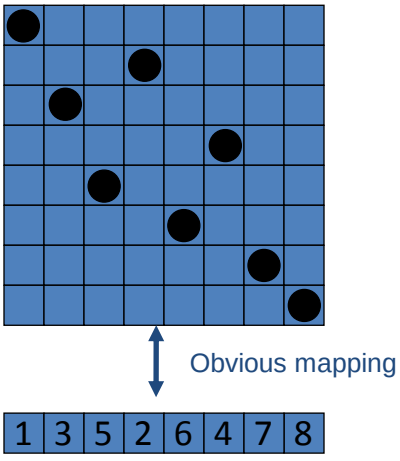
- Perform swap mutation on the two offspring produced in the previous slide.

Offspring 1	D C B F H E G A
Offspring 2	F H A D C E G B

- For Offspring 1, swap 2nd and 4th gene.
- For Offspring 2, swap 2nd and 6th gene.

Offspring 1	D F B C H E G A	55
Offspring 2	F E A D C H G B	40

The 8-Queen Problem: Representation



The 8-Queen Problem: Fitness Evaluation

- Penalty of one queen:
  the number of queens she can check.
- Penalty of a configuration:
  the sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration:
  inverse penalty to be maximized

The 8-Queen Problem: Mutation

- Small variation in one permutation, e.g.:
  - swapping values of two randomly chosen positions

The 8-Queens Problem: Crossover

- Combining two permutations into two new permutations:
  - choose random crossover point
  - copy first parts into children
  - create second part by inserting values from other parent:
    - in the order they appear there
    - beginning after crossover point
    - skipping values already in child