Introduction to Artificial Intelligence

Evolutionary Computation

Ing. Tomas Borovicka

Department of Theoretical Computer Science (KTI), Faculty of Information Technology (FIT)
Czech Technical University in Prague (CVUT)

BIE-ZUM, LS 2014/15, 5. lecture

https://edux.fit.cvut.cz/courses/BIE-ZUM/

Summary of Previous Lecture

- We introduced local search algorithms.
  - Operates using a single current configuration (candidate solution) and search in the neighborhood for better configuration to move in.
- We defined optimization problem.
  - The goal is to find the best configuration according to some objective function, path is irrelevant.
- Hill-climbing
  - Continually moves in the direction of increasing objective function.
- Simulated annealing
  - Combination of random hill-climbing search and Metropolis algorithm allows to move with certain probability to worse than current solution.
- Tabu search
  - Uses memory structures to maintain a history and prevent recent states being revisited.

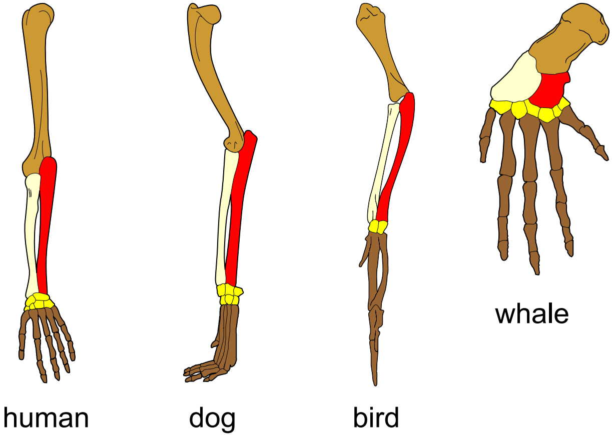
Evolutionary Computation

- Family of global meta-heuristic or stochastic optimization methods.
- Algorithms typically imitate some principle of natural evolution as method to solve optimization problems, e.g:
  - natural selection, survival of the fittest (Charles Darwin),
  - theory of genetic inheritance (Gregor Johann Mendel).
- Iteratively improves, ”breeds”, population of candidate solutions by selecting a recombining good quality candidates.
- Typically applied for black box problems where optimization is expensive.

Evolutionary Computation Techniques

- Gene expression programming
- Genetic algorithm
- Genetic/Evolutionary programming
- Evolution strategy
- Swarm intelligence
  - Ant colony optimization
  - Particle swarm optimization
  - Bees algorithm
  - Artificial immune systems
- Differential evolution
- Cultural algorithm
- Harmony search
- and many others...

Evolution of Chordate phylum

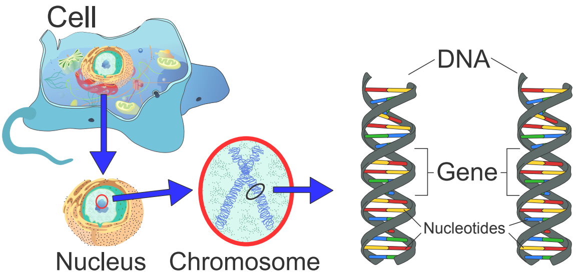


Practical use of Evolutionary Computation Techniques

- computer programs, functions fitting,
- automotive design, racing cars,
- robotics, design and behavior,
- hardware design, electronic circuits,
- computer gaming,
- encryption, code breaking,
- molecular design, chemistry and medicine,
- finance and investment strategies,
- neural networks design,
- ...

Biological Terminology

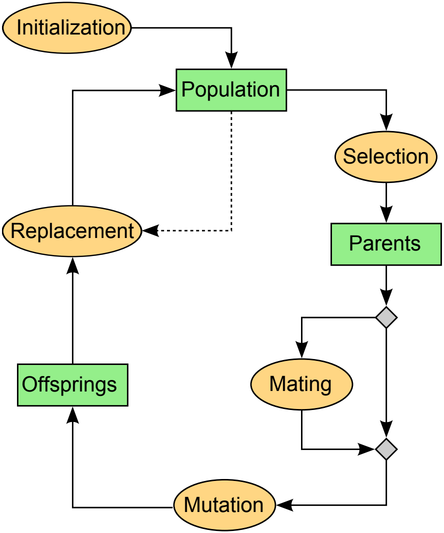
- Living organisms consist of cells. Cell's nucleus contains **chromosomes** that encode its DNA.
- A chromosome can be conceptually divided into **genes**, stretches of DNA encoding some trait, such as eye color, height etc.
- The different possible configurations for a trait are called **alleles** (e.g. blue, brown).
- Each gene is located at a particular position on the chromosome called **locus**.



Biological Terminology

- The complete collection of genetic material is called organism's **genome**.
- **Genotype** refers to a particular set of genes contained in a genome. Two individuals with identical genomes are said to have the same genotype.
- **Phenotype** is physical (or mental) characteristics of an organism (e.g. eye color, height, intelligence).
- **Recombination** (or **crossover**) is sexual reproduction when genes between two **parent** chromosomes are exchanged to form an **offspring**.
- **Mutation** is process in which single nucleotides (elementary bits of DNA) are changed from parent to offspring (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**)

General Evolutionary Computation Scheme



Evolutionary Computation Terminology

- The term **chromosome** refers to a candidate solution to a problem, typically encoded in a form of binary string.
- The **genes** are either single **bits** or short blocks of adjacent bits that encode a particular element of the candidate solution.
- The **genotype** of an individual is simply the configuration that individual's chromosome. It is a **grammar** of a solution.
- There is often no notion for **phenotype** in GA. We can see it as a **semantics** of a solution.
- An **allele** is one letter of chosen alphabet (e.g. 0 or 1 for bit)
- **Mutation** randomly changes the allele values of some locations in the chromosome (e.g. flipping random bit in binary string).
- **Crossover** is a process of exchanging genetic material between two single chromosome parents, i.e. exchanging part of configuration.
- Population in each iteration is called a **generation**.
- The entire set of generations is called a **run**.

Genetic Algorithm

- Invented at University of Michigan by John Holland in 1960s.
- Probably the most known evolutionary algorithm.
- From the beginning the goal was to formally study the phenomenon of natural adaptation and ability of general incorporation in problem solving.
- Result is universal "black box solver" for optimization using binary strings.

Genetic algorithm

Let $\mathbf{x} \in \{0, 1\}^n$ be a binary string (chromosome) of a length n and let $f(\mathbf{x})$ be a fitness function such that

$$f : \{0, 1\}^n \to \mathbb{R},$$

genetic algorithm solves an optimization problem

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \{0,1\}^n} (f(\mathbf{x})).$$

Genetic Algorithm

1. INITIALIZATION generate initialization population.
2. Calculate the fitness of each chromosome in the population.
3. Repeat the following steps until n offsprings have been created:
   (I) SELECTION Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness.
   (II) CROSSOVER With probability p_c (the "crossover probability" or "crossover rate"), cross over the pair to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.
   (III) MUTATION Mutate the offspring at each locus with probability p_m (the mutation probability or mutation rate), and place the resulting chromosomes in the new population.
4. Replace the current population with the new population.
5. Check stopping criteria (or convergence) and
   - terminate, or
   - go to step 2.

Population Initialization

Non-informed initialization

- Generates population of binary vectors randomly.

Informed initialization

- Population is generated with some knowledge.
  - Using simple heuristics.
  - Seeding with known good quality individuals.
- Risk that whole population will be placed in local optima that will be difficult or impossible to leave using crossover and mutation.

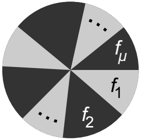
Selection

- Selection operators chooses good quality chromosomes from the population to be reproduced. The fitter the chromosome, the more times it is likely to be selected to reproduce.
- **Elitism** is an addition to selection operator, which forces to retain some number of best individuals of each generation.

Roulette-wheel selection

- Equivalent to giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness.

$$P_i(x) = \frac{f_i(x)}{\sum_{j=1}^{\mu} f_j(x)}$$



Tournament selection

- Randomly pick a small subset of chromosomes from the population and the chromosome with highest fitness becomes a parent.
- Not need to sort population by fitness!

Crossover

- Crossover operator combine or exchanges subparts of two chromosomes.
- Depends greatly on the problem and the encoding strategy.
- For some problems (with specific encoding) is even not used and evolution depends only on mutation.

Parent #1	1	0	0	1	1	0	1	1	1	0	0	1
Parent #2	1	0	1	1	1	0	0	1	1	1	0	0
Offspring #1	1	0	0	1	1	0	0	1	1	1	0	0
Offspring #2	1	0	1	1	1	0	1	1	1	0	0	1

One-Point Crossover

Parent #1	1	0	0	1	1	0	1	1	1	0	0	1
Parent #2	1	0	1	1	1	0	0	1	1	1	0	0
Offspring #1	1	0	0	1	1	0	0	1	1	1	0	1
Offspring #2	1	0	1	1	1	0	1	1	1	0	0	0

Two-Point Crossover

Parent #1	1	0	0	1	1	0	1	1	1	0	0	1
Parent #2	1	0	1	1	1	0	0	1	1	1	0	0
Offspring #1	1	0	1	1	1	0	1	1	1	1	0	1
Offspring #2	1	0	0	1	1	0	0	1	1	0	0	0

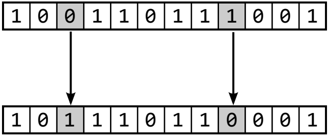
***n*-Point Crossover**

Parent #1	1	0	0	1	1	0	1	1	1	0	0	1
Parent #2	1	0	1	1	1	0	0	1	1	1	0	0
Offspring #1	1	0	1	1	1	0	1	1	1	0	0	1
Offspring #2	1	0	0	1	1	0	0	1	1	1	0	0

Uniform Crossover

Mutation

- Mutation operator randomly flips some of the bits in a chromosome.
- Mutation can occur at each bit position in a string with some probability p_m, usually very small (e.g., 0.001).



Algorithm 1 bitflip_mutation($\mathbf{x}$)

for $i \leftarrow 1$ **to** n **do**
    if random($\langle 0, 1 \rangle$) $< p_m$ **then**
        $\mathbf{x}[i] \leftarrow \neg \mathbf{x}[i]$
    end if
end for

Genetic Algorithm Pseudocode

Algorithm 2 Genetic Algorithm (GA)

```
P ← {}
for i ← 1 to μ do
    x ← random_individual()
    P ← P ∪ {(x, f(x))}
end for
repeat
    O ← {}
    for i ← 1 to μ/2 do
        p1 ← selection(P)
        p2 ← selection(P)
        (o1, o2) ← crossover(p1, p2)
        õ1 ← mutate(o1)
        õ2 ← mutate(o2)
        O ← O ∪ {(õ1, f(õ1)), (õ2, f(õ2))}
    end for
    P ← O
until termination condition is met
```

Parameters of Genetic Algorithm

- Initial population.
- Size of the population.
- Mutation rate.
- Crossover rate.
- Number of generations.

Moreover we have to decide about

- Encoding strategy.
- Fitness function.
- Crossover operators.
- Elitism.
- etc. . .

Knapsack Problem

Given

- knapsack with weight limit W,
- set of items $T = \{t_1, t_2, \ldots, t_n\}$ with weight $w(t_i)$ and value $v(t_i)$.

The goal is to find a subset $Z \subseteq T$ with maximal possible value so that the total weight is less than or equal to a given limit

$$\arg\max_{Z \subseteq T} \sum_{z \in Z} v(z) \quad \text{t.ž.} \quad \sum_{z \in Z} w(z) \leq W.$$

The knapsack problem is **NP-hard**, it means it does not exist an algorithm that solves the problem in polynomial time.

However, we can find near-optimal solution with genetic algorithm.

Knapsack Problem

- A chromosome can be represented in a binary string (array) having size equal to the number of the items.
- Each element from this array denotes whether an item is included in the knapsack ("1") or not ("0").
- Trivial fitness:

$$f(\mathbf{x}) = \begin{cases} \sum_{z\in\{t_j|x_j=1\}} v(z), & \text{pokud } \sum_{z\in\{t_j|x_j=1\}} w(z) \leq W, \\ 0, & \text{pokud } \sum_{z\in\{t_j|x_j=1\}} w(z) > W \end{cases}$$

- Can we do it better?

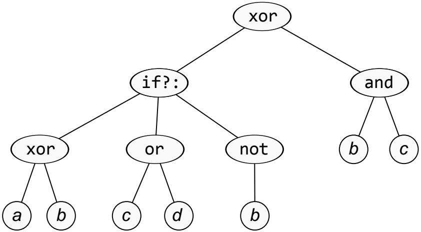
Genetic Programming

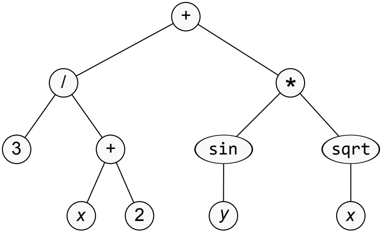
- Invented at Sanford University by J.R. Koza in 1960s.
- Motivation was general automatic programming.
- First use of genetic programming was to evolve Lisp programs to perform various tasks.
- More flexible then fixed chromosome, size of instance is part of evolution.
- Genotype is represented in a form of syntax tree.

Syntax Tree

Syntax tree consists of :

- **terminals** (leafs) – *T*
  - inputs of program, independent variables
- **functions** (internal nodes) – *F*
  - e.g. arithmetic operations, algebraic functions, logic functions etc.
- The sets of available functions and terminals form the **primitive set** of the genetic programming system.



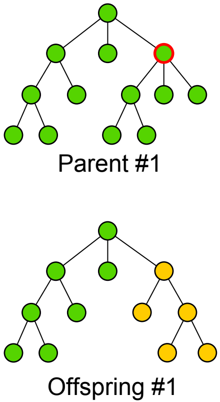


Population Initialization

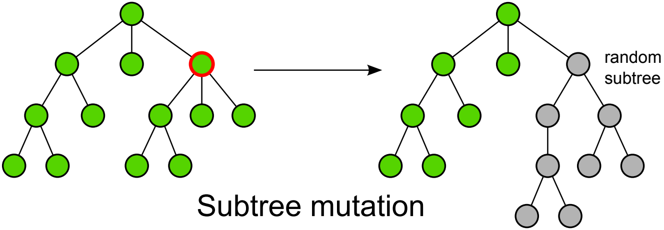
- Typically, individuals in the initial population are generated randomly.
- However, since it is not so trivial (such as generate random binary string) number of different approaches exist.
- First defined (and also simplest) methods are:
  - **GROW**
    - generated tree has depth $d \leq D_{max}$,
    - nodes in depth $d < D_{max}$ are randomly chosen from $F \cup T$,
    - nodes in depth $d = D_{max}$ are randomly chosen from T.
  - **FULL**
    - generated tree has depth $d = D_{max}$,
    - nodes in depth $d < D_{max}$ are randomly chosen from F,
    - nodes in depth $d = D_{max}$ are randomly chosen from T.
  - **Ramped half-and-half**
    - half of the initial population is constructed using full and half using grow.
  - **PTC1/2** (Probabilistic Tree-Creation)
    - generates tree with user defined expected size E_{tree},
    - to each function $f_i \in F$ and terminal $t_i \in T$ are assigned probabilities p_{f_i} to be chosen f_i when function is needed respectively p_{t_i} to be chosen t_i when terminal is needed

Subtree Crossover

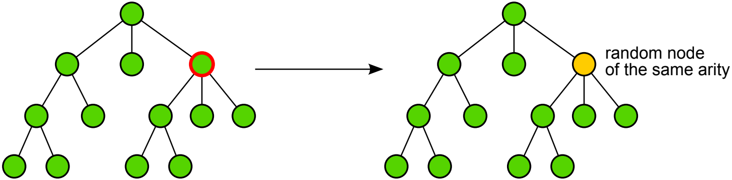
- Given two parents, subtree crossover randomly selects a crossover point in each parent tree.
- Two offsprings are created by replacing the subtrees rooted at the crossover points of both parents.



Mutation

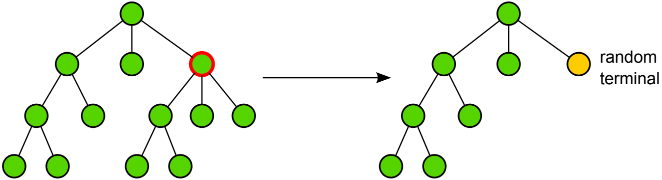


Subtree mutation

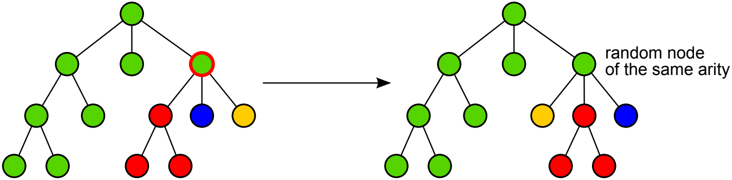


Point mutation

Mutation



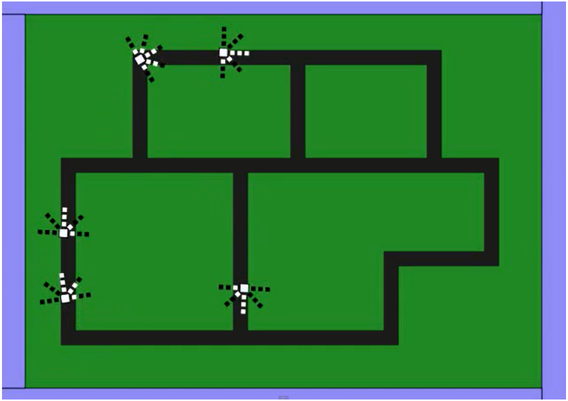
Shrink mutation



Permutation mutation

Example

Goal: Evolve robots driving on roads. Fitness is the average speed.



http://www.youtube.com/watch?v=lmPJeKRs8gE

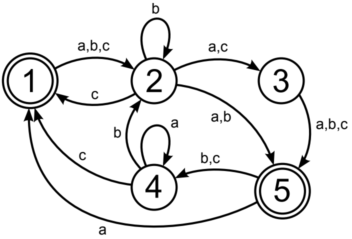
Evolutionary Programming

- Invented at University of California by L.J.Fogel in 1960s.
- Motivation was to generate alternative approach to artificial intelligence.
- One population of solutions, reproduction is only by mutation.
- Early versions of EP applied to the evolution of transition table of finite state machines.

Finite State Machine

State machine is described by:

- initial and final state,
- set of states,
- transition table.

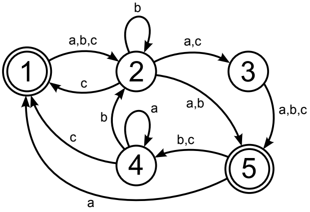


	a	b	c
1	2	2	2
2	3	2	3
3	5	5	5
4	4	2	1
5	1	4	4

Mutation

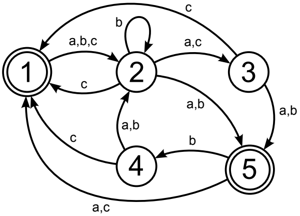
Five possible modes of mutation of state machine:

- add a state,
- delete a state,
- change the start state,
- change an output symbol,
- change a state transition.



	a	b	c
1	2	2	2
2	3	2	3
3	5	5	5
4	4	2	1
5	1	4	4

→



	a	b	c
1	2	2	2
2	3	2	3
3	5	5	1
4	2	2	1
5	1	4	1

Evolution Strategies

- Invented at Technische Universität Berlin by I. Rechenberg and H.P. Schwefel in 1960s.
- Based on the concept of the ”evolution of evolution”.
- Each individual is represented by its genotype and strategy parameters.
- Both genotype and strategy parameters are evolved.
- Mutated individuals are only accepted if fitness of parent is improved.

Selection

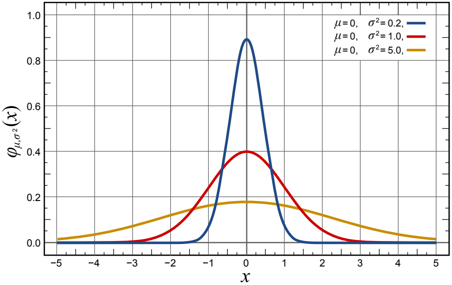
- First version $(1 + \lambda)$-ES,
  - one parent generates λ offsprings,
  - the best of the offsprings becomes parent of next generation,
  - note the similarity with hill-climbing!
- Advanced version $(\mu + \lambda)$-ES and (μ, λ)-ES
  - μ parents generates λ offsprings,
  - $(\mu + \lambda)$-ES selects from these $\mu + \lambda$ individuals μ best to the next generation.
  - (μ, λ)-ES selects to the next generation μ best only from offsprings.

Mutation

- **Gaussian mutation** - to each individual is added Gaussian distributed noise.

$$\mathbf{x}'(t) = \mathbf{x}(t) + N(0, \sigma(t))$$

- Adaptation of strategy parameters
  - Based on the $1/5$ success rule
    - σ is increased if the relative frequency of successful mutations over a certain period is larger than $1/5$
    - otherwise σ is decreased



Self Adaptation

- Each individual is represented by its genotype and strategy parameters

$$\mathcal{X}(t) = (\mathbf{x}(t), \sigma(t))$$

  $x(t) \in \mathbb{R}^n$ represents the genotype,
  σ represents the deviation strategy parameter vector.
- Both genotype and strategy parameters are evolved.
- Strategy parameters are self-adapted to determine
  - best search direction, and
  - maximum step size per dimension.

Crossover

- In the first version offspring was generated only through mutation.
- Newer versions defined crossover.
  - **Discrete** – two parents are randomly selected and recombined using discrete, multipoint crossover.
  - **Intermediate** – the offspring is a weighted average of the parents.
  - **Arithmetic** – the offspring is arithmetic average of the parents.
- Extension $(\mu/\rho \overset{+}{,} \lambda)$-ES indicates that ρ parents are used per application of the crossover operator,
  - $\rho = 1$ ...standard $(\mu \overset{+}{,} \lambda)$-ES,
  - $\rho = 2$ ...local crossover,
  - $\rho < \rho \leq \mu$ ...global crossover.