A0M33EOA
Genetic and Evolutionary Algorithms.

Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Heavilly using slides from Jiří Kubalík, CIIRC CTU, with permission.

Evolutionary Algorithms

Evolutionary algorithms: stochastic, population-based optimization algorithms.

- Stochastic, but not random search.
- Population-based counterpart to single-state local search methods (EAs are more robust w.r.t. getting stuck in local optima).
- Inspired by
  - Darwin's theory of evolution (random changes of individuals, and survival of the fittest).
  - Mendel's theory of inheritance (transfer of traits from parents to children), and
- They are not fast (black-box method, population-based).
- They are robust (efficient in finding good solutions in difficult search spaces).

Difference from a mere parallel hill-climber: candidate solutions affect the search of other candidates.

Originally, several distinct kinds of EAs existed:

- **Evolutionary programming, EP** (Fogel, 1966): real numbers, state automatons
- **Evolutionary strategies, ES** (Rechenberg, Schwefel, 1973): real numbers
- **Genetic algorithms, GA** (Holland, 1975): binary or finite discrete representation
- **Genetic programming, GP** (Cramer, Koza, 1989): trees, programs

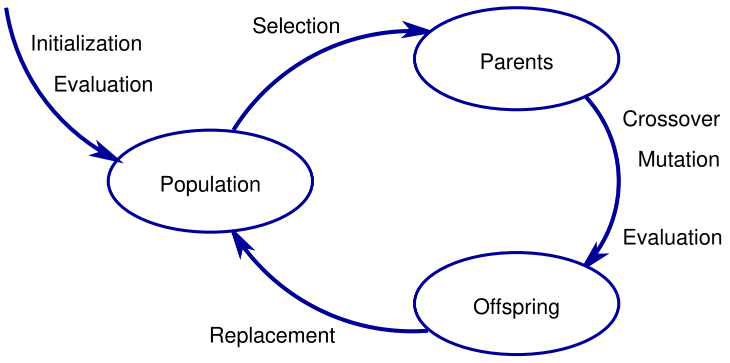
Currently, the focus is on emphasizing what they have in common, and on exchange of ideas among them.

Inspiration by biology

Vocabulary borrowed from natural genetics:

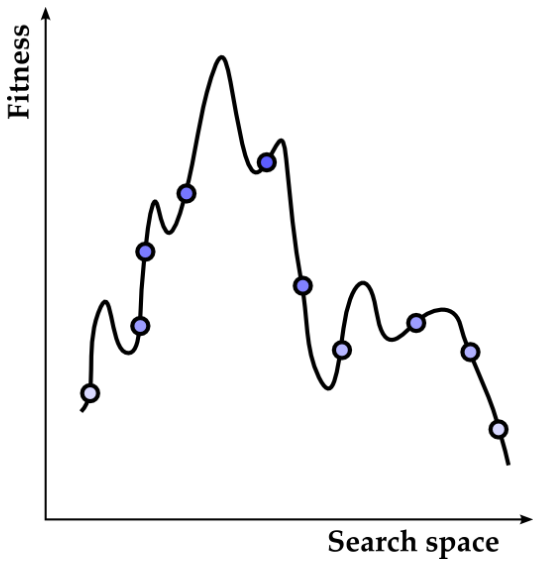
individual, chromosome	an encoded candidate solution
gene	a variable or a set of variables in a chromosome
locus	a place in a chromosome
allele	a particular value of gene at certain locus
fitness	quality of an individual
fitness function (landscape)	objective function
genotype	an individual's data structure as used during breeding
phenotype	the meaning of genotype, interpretation of the genotype by the fitness function
population	a set of candidate solutions
reproduction, selection	picking individuals based on their fitness
parents	individuals chosen by selection as sources of genetic material
children (offspring)	new individuals created by breeding
breeding	the process of creating children from a population of parents
mutation	perturbation of an individual; asexual breeding
recombination, crossover	producing one or more children from two or more parents; sexual breeding
generation	one cycle of fitness assessment, breeding, and replacement

Evolutionary cycle

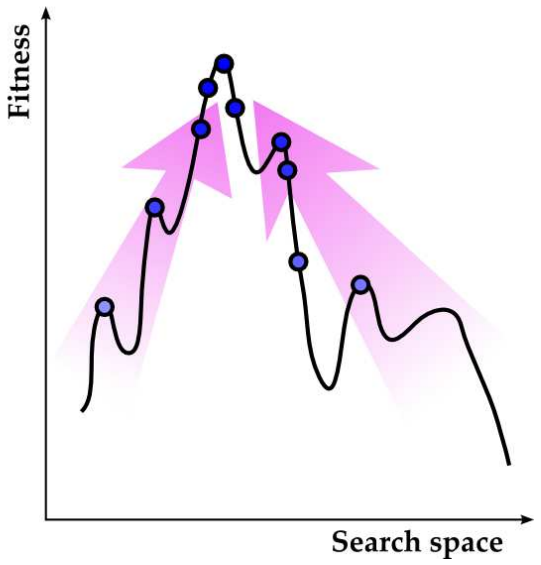


Idealized Illustration of Evolution

Uniformly sampled population



Population converging to promising regions



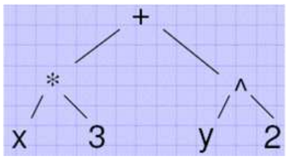
Representation

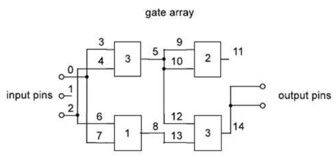
In conventional GAs, the solution is represented by

- a binary string: 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1

However, other types of representation can be more suitable for the problem at hand:

- real-valued string: 3,24 1,78 -2,61
- string of chars: D→E→A→C→B
- tree or graph:





- . . .

Evaluation Function

Objective (Fitness) function

- the only information about the sought solution the algorithm is endowed with,
- must be defined for every possible chromosome.

Fitness function may be

- nonlinear,
- multimodal,
- discrete,
- noisy,
- multidimensional,
- multiobjective.

Fitness does not have to be defined analytically:

- simulation/experiment results
- classification success rate
- human evaluation
- . . .

Fitness function should not be too costly (if possible)!!!

Example: Represenation & Evaluation

Function optimization

- maximization of $f(x, y) = x^2 + y^2$,
- parameters x and y take on values from interval $< 0, 31 >$,
- and are encoded on 5 bits each.

genotype	phenotype	fitness
0 0 0 0 0, 0 1 0 1 0	**0, 10**	**100**
0 0 0 0 1, 1 1 0 0 1	**1, 25**	**625 + 1 = 626**
0 1 0 1 1, 0 0 0 1 1	**11, 3**	**121 + 9 = 130**
1 1 0 1 1, 1 0 0 1 0	**27, 18**	**729 + 324 = 1053**

Quiz: Which binary chromosome represents the optimum in the above problem?

A `0 0 0 0 0 0 0 0 0 0`

B `1 1 1 1 1 1 1 1 1 1`

C None of the two.

D The function does not have any optimum.

Algorithm

Algorithm 1: Evolutionary Algorithm

```
begin
    $X$ ← InitializePopulation()
    $f$ ← Evaluate($X$)
    $x_{BSF}, f_{BSF}$ ← UpdateBSF($X, f$)
    while not TerminationCondition() do
        $X_N$ ← Breed($X, f$)                          // using certain breeding pipeline
        $f_N$ ← Evaluate($X_N$)
        $x_{BSF}, f_{BSF}$ ← UpdateBSF($X_N, f_N$)
        $X, f$ ← Join($X, f, X_N, f_N$)                // aka ‘‘replacement strategy’’
    return $x_{BSF}, f_{BSF}$
```

BSF : Best So Far

Algorithm 2: Canonical GA Breeding Pipeline

```
begin
    $X_S$ ← SelectParents($X, f$)
    $X_N$ ← Crossover($X_S$)
    $X_N$ ← Mutate($X_N$)
    return $X_N$
```

Other different `Breed()` pipelines can be pluged in the EA.

Initialization

Initialization is a process of creating individuals from which the search shall start.

- **Random:**
  - No prior knowledge about the characteristics of the final solution.
  - No part of the search space is preferred.
- **Informed:**
  - Requires prior knowledge about where in the search space the solution can be.
  - You can directly *seed* (part of) the population by solutions you already have.
- **Pre-optimization:**
  - (Some of) the population members can be set to the results of several (probably short) runs of other optimization algorithms.

Both *informed initialization* and *pre-optimization* introduce a *bias* into the search process:

+ they may help to find better solutions,
+ they may speed up the search process, but
– they may cause irreversible focus of the search process to regions with local optima!

Selection

Selection is the process of choosing which population members shall become parents.

- Usually, the better the individual, the higher chance of being chosen.
- A single individual may be chosen more than once; better individuals influence more children.

Selection types:

- **No selection:** all population members become parents (and the selection pressure must be applied in replacement).
- **Uniform selection:** each population member has the same chance of becoming a parent (and the selection pressure must be applied in replacement).
- **Fitness-proportional selection:** the probability of being chosen is proportional to the individual's fitness.
- **Rank-based selection:** the probability of being chosen is proportional to the rank of the individual in population (when sorted by fitness).
- **Tournament selection:** the set of parents is composed of the winners of small tournaments (choose n individuals uniformly, and pass the best of them as one of the parent).
- **Truncation selection:** the best n % of the population become parents.
- . . .

Mutation

Mutation makes small changes to the population members (usually, it iteratively applies *perturbation* to each individual). It

- promotes the population diversity,
- minimizes the chance of loosing a useful part of genetic code, and
- performs a local search around individuals.

Selection + mutation:

- Even this mere combination may be a powerfull optimizer.
- It differs from several local optimizers run in parallel.

Types of mutation:

- For binary representations: bit-flip mutation
- For vectors of real numbers: Gaussian mutation, . . .
- For permutations: 1-opt, 2-opt, . . .
- . . .

Crossover

Crossover (xover) combines the traits of 2 or more chosen parents.

- Hypothesis: by combining features of 2 (or more) good individuals we can maybe get even better solution.
- Crossover usually creates children in unexplored parts of the search space, i.e., promotes diversity.

Types of crossover:

- For vector representations: 1-point, 2-point, uniform
- For vectors of real numbers: geometric xover, simulated binary xover, parent-centric xover, . . .
- For permutations: partially matched xover, edge-recombination xover, . . .
- . . .

Replacement

Replacement strategy (the `join()` operation) implements the *survival of the fittest* principle. It determines which of the members of the old population and which new children shall survive to the next generation.

Types of replacement strategies:

- **Generational:** the old population is thrown away, new population is chosen just from the children.
- **Steady-state:** members of the old population joined with the newly created offspring; any of them may survive to the next generation.
- Similar principles as for selection can be applied.

Why EAs?

EAs are popular because they are

- easy to implement,
- robust w.r.t. problem formulations, and
- less likely to end up in a local optimum.

Some of the application areas:

- control,
- engineering design,
- image processing,
- planning & scheduling,
- VLSI circuit design,
- network optimization & routing problems,
- optimal resource allocation,
- marketing,
- credit scoring & risk assessment,
- and many others.

John Holland: *"It's best used in areas where you don't really have a good idea what the solution might be. And it often surprises you with what they come up with."*

Reproduction

Reproduction (parental selection) models nature's survival-of-fittest principle

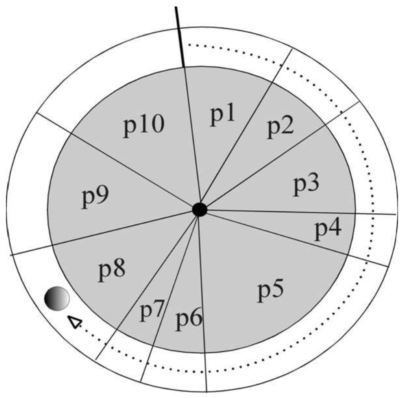
- prefers better individuals to the worse ones,
- still, every individual should have a chance to reproduce.

Roulette wheel:

- Prob. of choosing a solution is directly proportional to its fitness value:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i},$$

  where N is population size.
- To select n individuals, you need to spin the wheel n times.



Quiz: If an individual i has a selection probability p_i, how many of its copies will be present in the set of N parents:

A 1

B N

C $\lfloor p_i \cdot N \rfloor$

D Any number between 0 and N

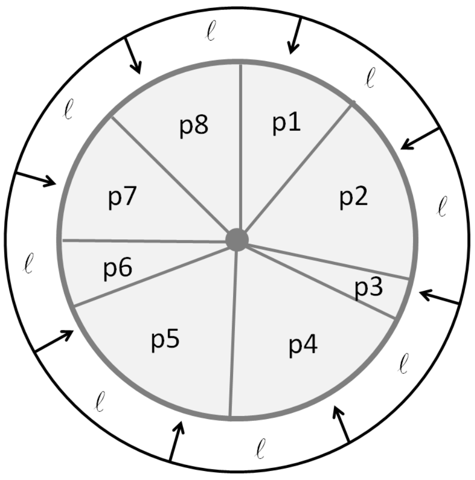
Expected frequencies vs. observed frequencies

- *Expected selection frequency*: given selection probability of i-th individual, p_i, and N individuals to be selected, we expect to get on average $N \cdot p_i$ copies of individual i.
- *Observed selection frequency*: can be anywhere between 0 and N.

Stochastic Universal Sampling

SUS ensures that the observed selection frequencies of each individual are in line with the expected frequencies:

- Extra wheel, let's denote it a *pointer wheel*, with equidistantly distributed N (pop. size) pointers.
  Ex.: If we are selecting 8 individuals, the pointer wheel will have 8 pointers distributed with $360/8 = 45$ degrees step size.
- SUS works by making a single spin of the pointer wheel.
- A single rotation of the pointer wheel selects all of N individuals at once.



- Every individual i receives a number of copies from interval $(\lfloor N \cdot p_i \rfloor, \lceil N \cdot p_i \rceil)$.
  Ex.: If we have an individual that occupies 4.5% of the roulette wheel and we select 100 individuals, we would expect on average 4.5 copies for that individual to be selected. Then, the individual will be selected either four or five times. Neither more, nor less.

Reproduction: Premature Convergence & Stagnation

Two (strongly related) issues in the evolution process:

- **population diversity**,
- **selection pressure**.

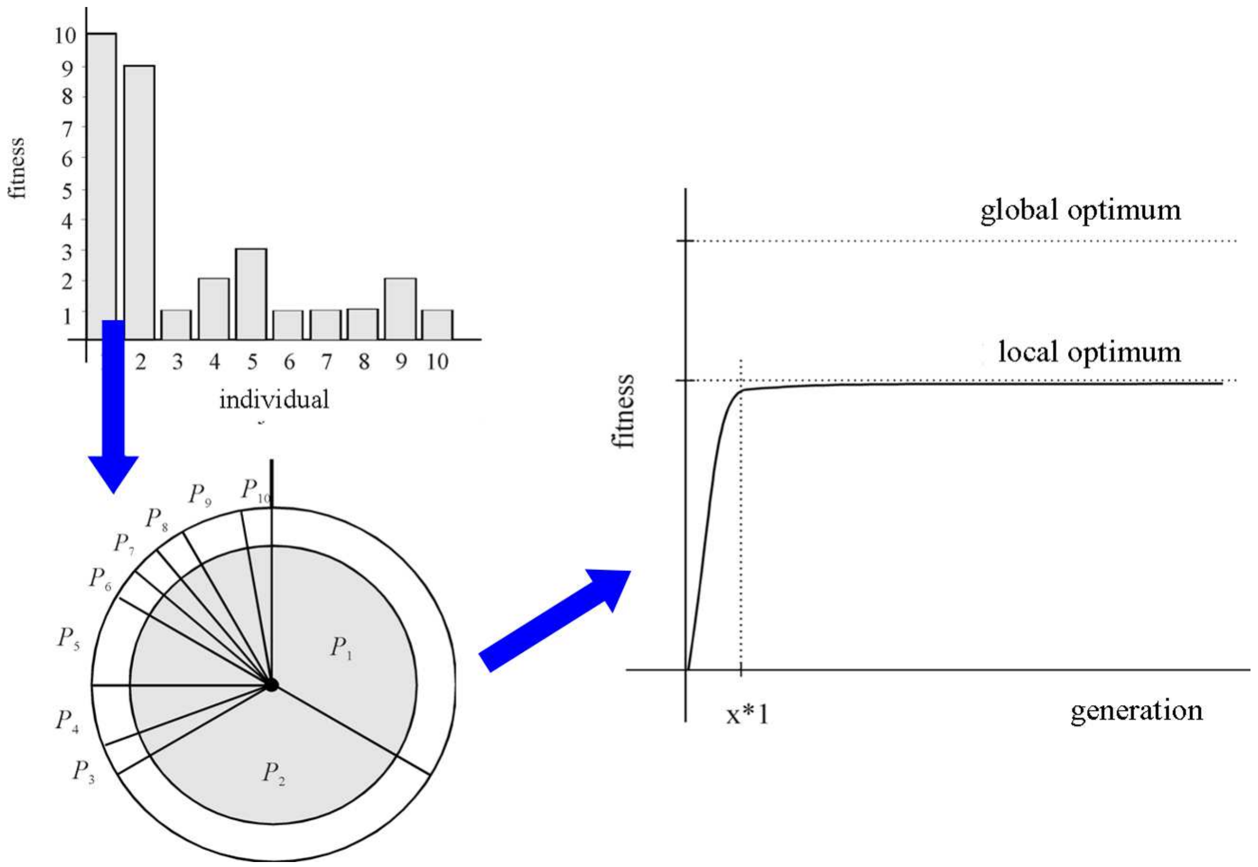
Premature convergence:

- A premature loss of diversity in the population with the search converging to a sub-optimal solution.
- Early stages of the evolution search process.

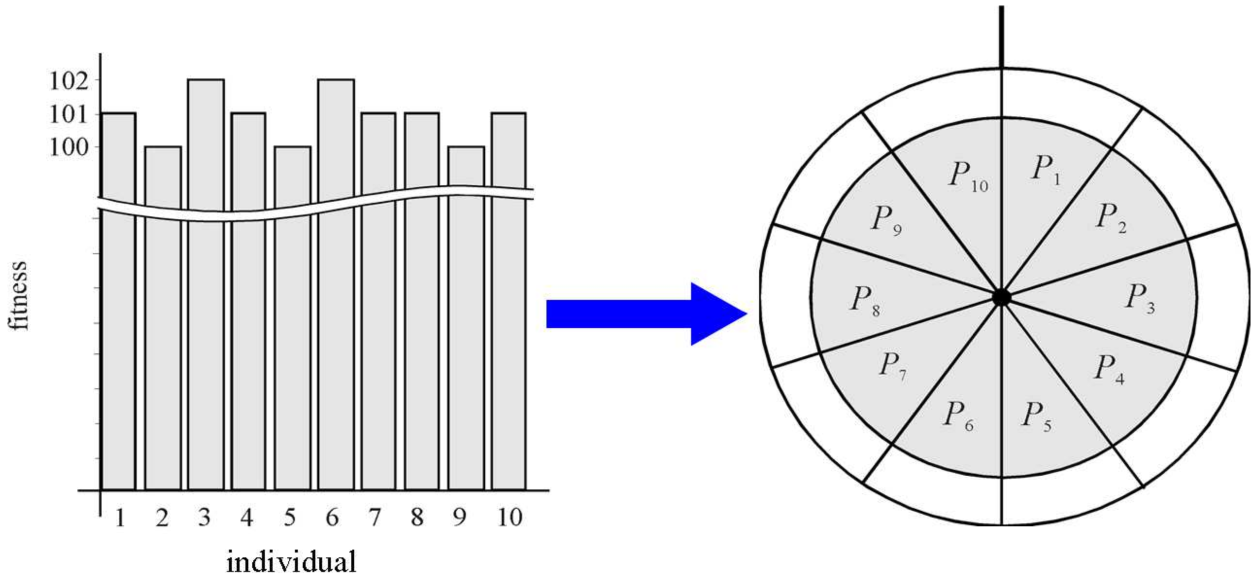
Stagnation:

- Ineffective search due to a weak selection pressure.
- Later stages of the evolution search process.

Premature Convergence



Stagnation



How to Deal with it?

Balance between **exploration** and **exploitation**.

- How to achieve the optimal selection pressure during the whole evolutionary search?

Options:

- scaling techniques,
- proper selection mechanisms,
- fitness sharing and crowding,
- ....

Scaling

Linear scaling: adjustment of the fitness values distribution in order to get the desired selection pressure

$$\sigma = f_{max}/f_{avg}$$

The actual chromosomes' fitness is scaled as

$$f'_i = a \cdot f_i + b$$

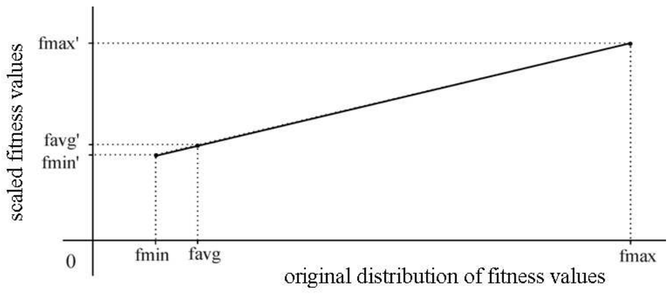
Parameters a and b are usually determined so that

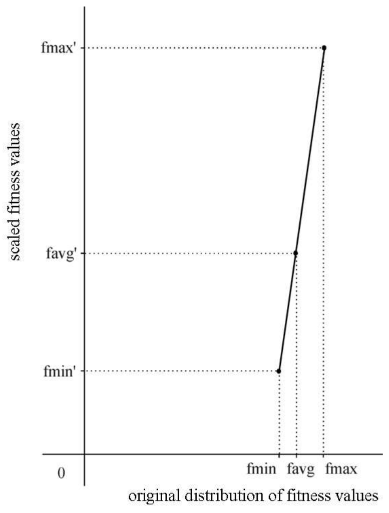
- the average fitness is mapped to itself, and
- the best fitness is a desired multiple of the average fitness.

Typical value of σ is from $(1.5, 2.0)$

Effect of Linear Scaling

Linear scaling helps to remedy both the premature convergence and stagnation.





Diversity Preservation: Fitness Sharing

Diversity preservation method proposed for solving multi-modal optimization problems so that GA is able to discover and evenly sample all optima.

Idea: decrease fitness of similar solutions

Algorithm to calculate the shared fitness value of i-th individual in population of size N

1. Calculate the distances d_{ij} of individual i to all individuals j.
2. Calculate values of *sharing function* between individual i and all individuals j:

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha}, \text{if } d_{ij} \le \sigma_{share}, \\ 0, \text{otherwise.} \end{cases}$$

3. Calculate *niche count* nc_i of individual i:

$$nc_i = \sum_{j=1}^{N} Sh(d_{ij})$$

4. Calculate *shared fitness* of individual i:

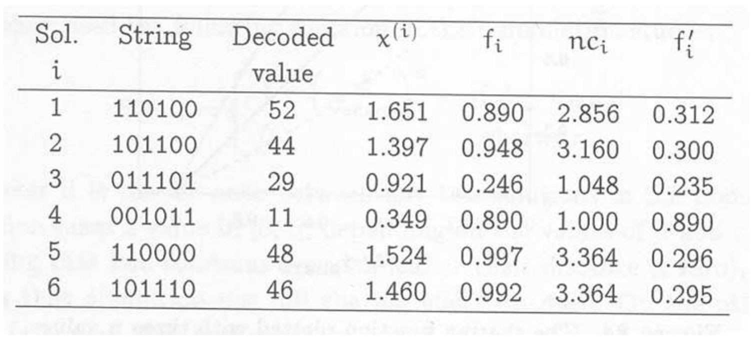
$$f'_i = f_i / nc_i$$

Remark: If $d = 0$, then $Sh(d) = 1$, meaning that two solutions are identical. If $d \ge \sigma_{share}$, then $Sh(d) = 0$ meaning that two solutions do not have any sharing effect on each other.

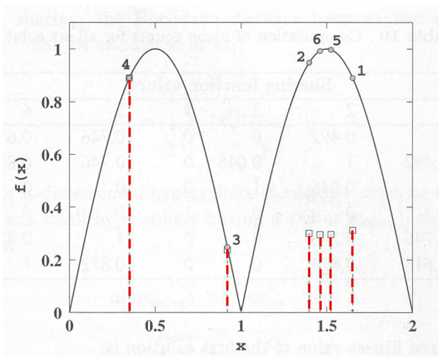
Fitness Sharing: Example

Bimodal function, six solutions and corresponding shared fitness functions

- $\sigma_{share} = 0.5$, $\alpha = 1$.



Sol. i	String	Decoded value	$x^{(i)}$	f_i	nc_i	f'_i
1	110100	52	1.651	0.890	2.856	0.312
2	101100	44	1.397	0.948	3.160	0.300
3	011101	29	0.921	0.246	1.048	0.235
4	001011	11	0.349	0.890	1.000	0.890
5	110000	48	1.524	0.997	3.364	0.296
6	101110	46	1.460	0.992	3.364	0.295



©Kalyanmoy Deb: Multi-Objective Optimization using Evolutionary Algorithms.

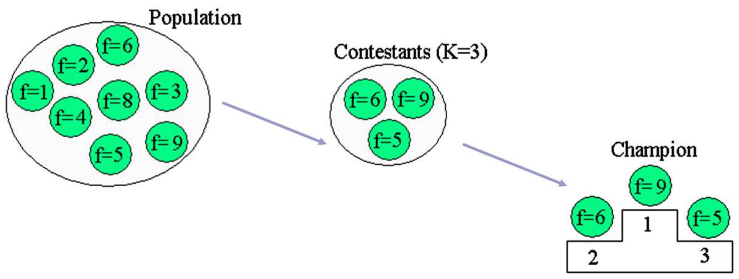
Let's calculate the shared fitness value of the first solution

- $d_{11} = 0.0$, $d_{12} = 0.254$, $d_{13} = 0.731$, $d_{14} = 1.302$, $d_{15} = 0.127$, $d_{16} = 0.191$
- $Sh(d_{11}) = 1$, $Sh(d_{12}) = 0.492$, $Sh(d_{13}) = 0$, $Sh(d_{14}) = 0$, $Sh(d_{15}) = 0.746$, $Sh(d_{16}) = 0.618$.
- $nc_1 = 1 + 0.492 + 0 + 0 + 0.746 + 0.618 = 2.856$
- $f'(1) = f(1)/nc_1 = 0.890/2.856 = 0.312$

Tournament Selection

Tournament selection: the best out of n randomly chosen individuals is selected.

- n is the size of the tournament.
- Rank-based method: the size of differences in fitness among individuals do not matter.
- To select N individuals, the tournament must be executed N times.



Genetic Operators

Crossover

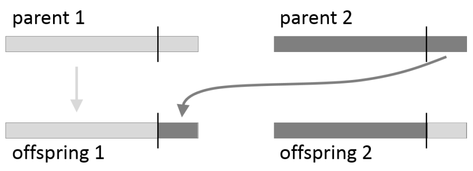
Role of crossover

- sampling (exploration) of the search space
- combining features of individuals

Idea (or hypothesis):

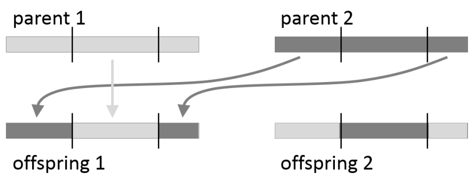
- Given two well-fit solutions to the given problem, it is possible to get a new solution that is even better than both its parents by properly mixing the two parents. (?)

1-point crossover



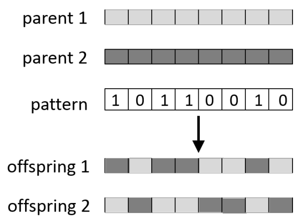
- Mixing of two parents determined by a **single crossing point** chosen randomly for each pair of parents.

2-point crossover



- Mixing of two parents determined by **two crossing points** chosen randomly for each pair of parents
- Higher exploration power than 1-point crossover.

Uniform crossover



- Mixing of two parents determined by an **n-point inheritance template** chosen randomly for each pair of parents.
- Highest exploration power.

Quiz: How many different offspring can be created with uniform crossover from two parents of length l, if they differ on d positions?

A 2

B 2^l

C 2^d

D 2^{l-d}

EAs and Optimization with Constraints

Search space with *constraints* contains both feasible and infeasible solutions.

Example: Assume the Traveling Salesman Problem and a simple one-point crossover. It is easy to get an infeasible solution, even when both parents are feasible.

$$\begin{array}{lcl} \text{parent1: } 4\,8\,1 \mid 3\,5\,2\,7\,6 & & \text{offspring1: } 4\,8\,1 \mid 1\,3\,6\,5\,4 \\ & \longrightarrow & \\ \text{parent2: } 2\,7\,8 \mid 1\,3\,6\,5\,4 & & \text{offspring2: } 2\,7\,8 \mid 3\,5\,2\,7\,6 \end{array}$$

Neither of the two offspring represents a feasible solution:

- some cities are missing,
- some cities are duplicated in the tour.

Solutions:

- repair mechanisms
- special operators

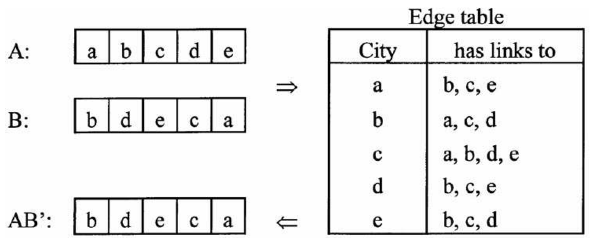
TSP: Edge-Recombination Operator

Direct representation

genotype: a e d b c
tour: a → e → d → b → c

Edge recombination crossover

- Create a table of neighbors (*edge table*) – for each city *i* there is a list of cities that have a link to *i* in the parental tours.
- Start creating a tour in a randomly chosen city, *currentCity*.
  Remove all occurrences of *currentCity* from the edge table.



Edge table

City	has links to
a	b, c, e
b	a, c, d
c	a, b, d, e
d	b, c, e
e	b, c, d

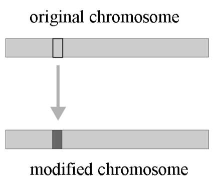
- Choose a new *currentCity* among the unused neighbors of *currentCity* in the edge table.
  If *currentCity* has already an empty list of unused neighbors, choose an arbitrary city that is not yet in the created tour.
  Remove all occurrences of *currentCity* from the edge table.
  Repeat this step until all cities have been added to the tour.

Mutation

Roles of mutation:

- preserves population diversity
- minimizes the chance of loosing some important piece of genetic information

Single bit-flip mutation event



Population with missing genetic information

```
0 0 1 1 0 0 0 1 1 0
0 1 1 0 0 1 0 1 0 0
0 0 0 1 1 0 1 0 1 1
0 1 0 0 1 0 0 1 1 1
0 1 1 0 0 0 0 1 0 1
        .   .   .
0 1 0 0 1 1 0 1 0 0
```

Usual implementation of mutation:

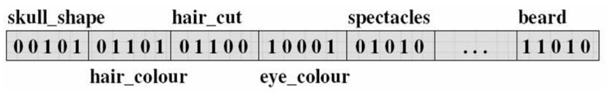
- For each bit independently, decide randomly whether it should be mutated or not.
- **Probability of mutation**: p_m
- Default value: $p_m = \frac{1}{l}$, where l is the chromosome length.
- The actual number of mutated bits in a chromosome is a random variable from $\langle 0, l \rangle$.
- The expected number of mutated bits is $p_m \cdot l$.

Criminal suspect identification

Database of "criminals"

Goal is to guide the witness through a huge database of faces to identify the criminal suspect.

Chromosome structure



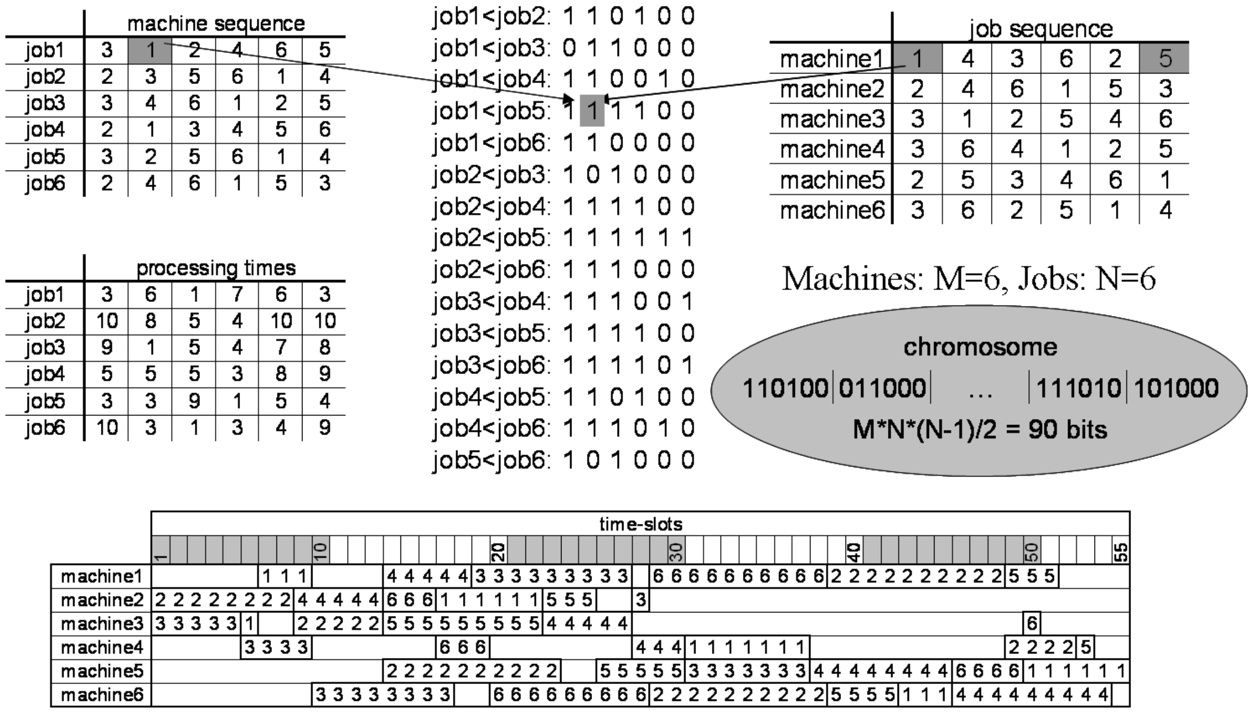
Interactive evolution:

- Human serves as an objective function evaluator!!!

? UK Home Office, Police Systems Research and Development Group ?

Job Shop Scheduling Problem

Representation: Pair-wise relative order of jobs on every machine



	machine sequence					
job1	3	1	2	4	6	5
job2	2	3	5	6	1	4
job3	3	4	6	1	2	5
job4	2	1	3	4	5	6
job5	3	2	5	6	1	4
job6	2	4	6	1	5	3

	processing times					
job1	3	6	1	7	6	3
job2	10	8	5	4	10	10
job3	9	1	5	4	7	8
job4	5	5	5	3	8	9
job5	3	3	9	1	5	4
job6	10	3	1	3	4	9

```
job1<job2: 1 1 0 1 0 0
job1<job3: 0 1 1 0 0 0
job1<job4: 1 1 0 0 1 0
job1<job5: 1 1 1 1 0 0
job1<job6: 1 1 0 0 0 0
job2<job3: 1 0 1 0 0 0
job2<job4: 1 1 1 1 0 0
job2<job5: 1 1 1 1 1 1
job2<job6: 1 1 1 0 0 0
job3<job4: 1 1 1 0 0 1
job3<job5: 1 1 1 1 0 0
job3<job6: 1 1 1 1 0 1
job4<job5: 1 1 0 1 0 0
job4<job6: 1 1 1 0 1 0
job5<job6: 1 0 1 0 0 0
```

	job sequence					
machine1	1	4	3	6	2	5
machine2	2	4	6	1	5	3
machine3	3	1	2	5	4	6
machine4	3	6	4	1	2	5
machine5	2	5	3	4	6	1
machine6	3	6	2	5	1	4

Machines: M=6, Jobs: N=6

chromosome

110100|011000| … |111010|101000

M*N*(N-1)/2 = 90 bits

	time-slots																																																					
	1									10										20										30										40										50					55
machine1							1	1	1					4	4	4	4	4	3	3	3	3	3	3	3	3	3		6	6	6	6	6	6	6	6	6	6	2	2	2	2	2	2	2	2	2	2	5	5	5				
machine2	2	2	2	2	2	2	2	2	4	4	4	4	4	6	6	6	1	1	1	1	1	1	5	5	5			3																											
machine3	3	3	3	3	3	1			2	2	2	2	2	5	5	5	5	5	5	5	5	5	4	4	4	4	4																							6					
machine4						3	3	3	3								6	6	6									4	4	4	1	1	1	1	1	1	1												2	2	2	2	5		
machine5														2	2	2	2	2	2	2	2	2	2			5	5	5	5	5	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	6	6	6	6	1	1	1	1	1	1
machine6										3	3	3	3	3	3	3	3			6	6	6	6	6	6	6	6	6	2	2	2	2	2	2	2	2	2	2	5	5	5	5	1	1	1	4	4	4	4	4	4	4	4	4	

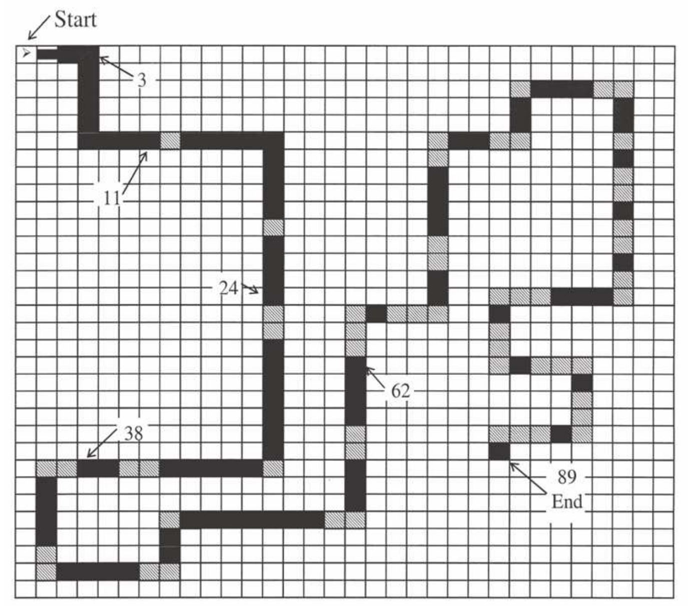
Artificial Ant Problem

Santa Fe trail

- 32 × 32 **grid** with 89 food pieces.
- **Obstacles**
  - 1×, 2× strait,
  - 1×, 2×, 3× right/left.

Ant capabilities

- **detects** the food right in front of him in direction he faces.
- **actions** observable from outside
  - MOVE – makes a step and eats a food piece if there is some,
  - LEFT – turns left,
  - RIGHT – turns right,
  - NO-OP – no operation.



Goal: find a strategy that would navigate an ant through the grid finding all the food pieces in the given time (600 time steps).

Artificial Ant Problem: GA Approach

Collins a Jefferson 1991, standard GA using binary representation

Representation:

- Strategy represented as a finite state machine.
- Transition table encoded as binary chromosome of fixed length.

Example: 4-state FSM, 34-bit long chromosomes ($2 + 4 \times 8$)

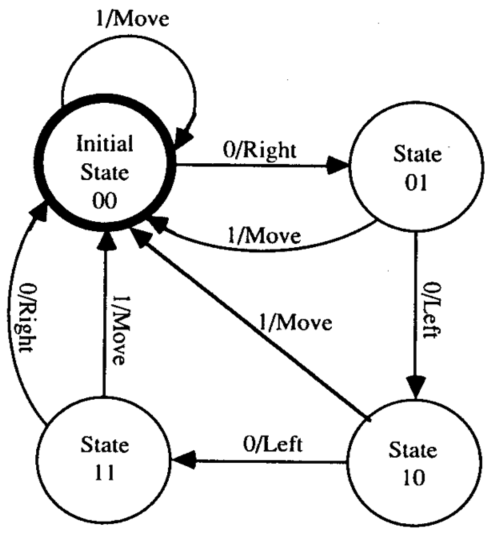
	Current state	Input	New state	Operation
1	00	0	01	10 = Right
2	00	1	00	11 = Move
3	01	0	10	01 = Left
4	01	1	00	11 = Move
5	10	0	11	01 = Left
6	10	1	00	11 = Move
7	11	0	00	10 = Right
8	11	1	00	11 = Move

00	0110	0011	1001	0011	1101	0011	0010	0011

Artificial Ant Problem: GA Result

Ant behavior

- What happens if the ant ″hits″ an obstacle?
- What is strange with transition from state 10 to the initial state 00?
- When does the ant succeed?
- Is the number of states sufficient to solve the problem?
- Do all of the possible 34-bit chromosomes represent a feasible solution?



GA result

- Representation:
  - 32 states
  - chromosome length: $453 = 64 \times 7 + 5$ bits !!!
- **Population size: 65.536 !!!**
- **Number of generations: 200**
- **Total number of samples tried:** 13×10^6 **!!!**

Schema Theory

Schema Theory

Schema theory (J. Holland, 1975):

- Analyzes effect of selection, crossover and mutation on the population's genotype
- Tries to answer the question: **″Why and How Evolutionary Algorithms Work?″**

In its original form the schema theory assumes:

- binary representation,
- proportionate roulette wheel selection,
- 1-point crossover and bit-flip mutation.

Schemata

Schema: a template, which defines a set of solutions with certain specific similarities.

- consists of 0s, 1s (fixed values) and wildcard symbols * (any value),
- covers 2^r strings, where r is a number of $*$ used in the schema.
  Example: schema $S =$ `11*0*` covers strings 11000, 11001, 11100, and 11101.

Schema properties

- **Defining length** $\delta(S)$ (compactness): distance between first and last non-* in a schema (= number of positions where 1-point crossover can disrupt the schema).
- **Order** $o(S)$ (specificity): a number of non-*'s (= number of positions where simple bit swapping mutation can disrupt the schema).
  - Chromosomes are order l schemata, where l is length of chromosome (in bits or loci).
  - Chromosomes are instances (or members) of lower-order schemata.
- **Fitness** $f(S)$ (quality): average fitness computed over all covered strings.
  Example: $S =$ `**1*01*0**`: $\delta(S) = 5$, $o(S) = 4$

Schemata: Example

8-bit Count Ones problem:
maximize a number of ones in 8-bit string.

string	fitness		string	fitness
00000000	0		11011111	7
00000001	1	. . .	10111111	7
00000010	1		01111111	7
00000100	1		11111111	8

Schema	$S_a =$ `1*1**10*`	$S_b =$ `*0*0****`
defining length	$\delta(S_a) = 7 - 1 = 6$	$\delta(S_b) = 4 - 2 = 2$
order	$o(S_a) = 4$	$o(S_b) = 2$
fitness	$f(S_a) = 5$	$f(S_b) = 3$

- S_a covers 1 string of fitness 3, 4 strings of fitness 4, 6 with fitness 5, 4 with withess 6, and 1 with fitness 7, i.e., $f(S_a) = (1 \cdot 3 + 4 \cdot 4 + 6 \cdot 5 + 4 \cdot 6 + 1 \cdot 7)/16 = 80/16 = 5$
- $f(S_b) = (1 \cdot 0 + 6 \cdot 1 + 15 \cdot 2 + 20 \cdot 3 + 15 \cdot 4 + 6 \cdot 5 + 1 \cdot 6)/2^6 = 192/64 = 3$

Quiz: What is the fitness of $S =$ `*0*1****` compared to S_b?

A S is worse than S_b, not clear by how much.

B S is worse than S_b by 1.

C S is better than S_b, not clear by how much.

D S is better than S_b, by 1.

Schema Theorem: Effect of Reproduction

Let $m(S,t)$ be a number of instances (strings) of schema S in population of size N at time t.

Question: How do schemata propagate? What is a lower bound on change in sampling rate of a single schema from generation t to $t+1$?

Effect of fitness-proportionate roulette wheel selection:

- A string a_i is copied according to its fitness; it gets selected with a probability

$$p_i = \frac{f_i}{\sum f_j}.$$

- After picking N strings with replacement from the population at time t, we expect to have $m(S,t+1)$ representatives of the schema S in the population at time $t+1$ as given by the equation

$$m(S,t+1) = N \cdot m(S,t) \cdot \frac{f(S)}{\sum f_j},$$

  where $f(S)$ is the fitness of schema S at time t.
- The formula can be rewritten as

$$m(S,t+1) = m(S,t) \cdot \frac{f(S)}{f_{avg}},$$

  where f_{avg} is the average fitness of the population.

Schema Theorem Derivation: Effect of Crossover and Mutation

Effect of 1-point Crossover:

- **Survival probability** p_s: let's make a conservative assumption that crossover within the defining length of S is always disruptive to S, and ignore gains.
- Crossover probability p_c: fraction of population that undergoes crossover.

$$p_s \geq 1 - (p_c \cdot \delta(S)/(L-1))$$

  Example: Compare survival probability of $S = (11****)$ and $S = (1****0)$.

Effect of Mutation:

- Each fixed bit of schema ($o(S)$ of them) changes with probability p_m, so they all stay unchanged with probability

$$p_s = (1-p_m)^{o(S)}.$$

- This can be approximated as

$$p_s = (1 - o(S) \cdot p_m),$$

  assuming $p_m \ll 1$.

Schema Theorem

Finally, we get a ”classical” form of the **reproductive schema growth equation**:

$$m(S,t+1) \geq m(S,t) \cdot \frac{f(S)}{f_{avg}} \cdot [1 - p_c \cdot \frac{\delta(S)}{L-1} - o(S) \cdot p_m].$$

What does it tell us?

Schema theorem: Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

Building Block Hypothesis: A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the **building blocks**.

David Goldberg: ”*Short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness: we construct better and better strings from the best partial solutions of the past samplings.*”

Y. Davidor: ”*The whole GA theory is based on the assumption that one can state something about the whole only by knowing its parts.*”

Corollary: Choosing the right representation/encoding is critical for GA performance; chosen encoding should satisfy the idea of short building blocks.

Summary

EA Materials: Reading, Demos, Software

Classic literature:

- D. E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
- Z. Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs, Springer, 1998.
- Z. Michalewicz: How to solve it? Modern heuristics. 2nd ed. Springer, 2004.

Demos

- M. Obitko: Introduction to genetic algorithms with java applets `https://www.obitko.com/tutorials/genetic-algorithms/`

Software

- DEAP: EC framework for Python
  `https://deap.readthedocs.io/en/master/`
- ECJ: A Java-based Evolutionary Computation Research System
  `http://cs.gmu.edu/~eclab/projects/ecj/`
- EO: Evolving Objects - EC library for C++
  `http://eodev.sourceforge.net/`
- . . .

Learning outcomes

After this lecture, a student shall be able to

- know and actively use the terminology inspired by biology;
- explain the importance, role and types of representation, selection, and genetic operators;
- distinguish between premature convergence and stagnation of EA, and suggest methods to fight them;
- explain the trade-off between exploration and expoitation;
- implement 1- and 2- point crossover, uniform crossover, bit-flip mutation;
- describe options for encoding the solutions of traveling salesperson problem, and the relevant genetic operators;
- give examples of real-world problems EAs have been applied to;
- describe what a schema theory is and what it is used for;
- explain the schema theorem, and the influence of its individual parts related to selection, crossover and mutation;
- state the so-called Building Block Hypothesis and comment on its relation to the chosen representation.