Evolutionary Computation

2025/26

Master Artificial Intelligence

Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

25/26

Evolutive methods

- Evolutive methods work with populations of individuals (or only one individual and a certain type of memory).
- There are probabilistic modification processes (mutation, reproduction, recombination/crossover) that change the population from one to the next generation.
- The performance of the individuals is based on a fitness which usually is the objective function (but not necessarily).
- There is a selection process to maintain a (more or less) stable state (size) of the population.
- Most of the algorithmic decisions are drawn probabilistically.

I will not give details on the history and researchers, please, take a look at the literature/bibliography.

Genetic algorithms (GA)

- We distinguish the genotype (codification of the individuals) and the phenotype (elements of the search space).
- There should exist a bijection between genotype and phenotype.
  (mapping of phenotype to genotype can be relaxed)
- The genotype encodes the free parameters of an individual.
- The modifications (mutation and recombination/crossover) are carried out over the genotype.
- The fitness is evaluated over the phenotype (our objective function).
- We have to explain: codification (of the genotype), initialization, mutation, recombination/crossover, selection, and stopping.

GA: principal loop

A genetic algorithm can be summarized in the following principal loop:

```
InitializePopulation()          # initialization
EvaluateIndividuals()           # evaluation
while not Stopping():           # stopping
  DetermineParents()              # selection
  GenerateChildren()              # recombination
  MutateChildren()                # mutation
  EvaluateIndividuals()           # evaluation
  ReestablishPopulation()         # selection
```

GA: encoding of the individuals

There are many possibilities how to encode the free parameters of an individual to form its genotype:

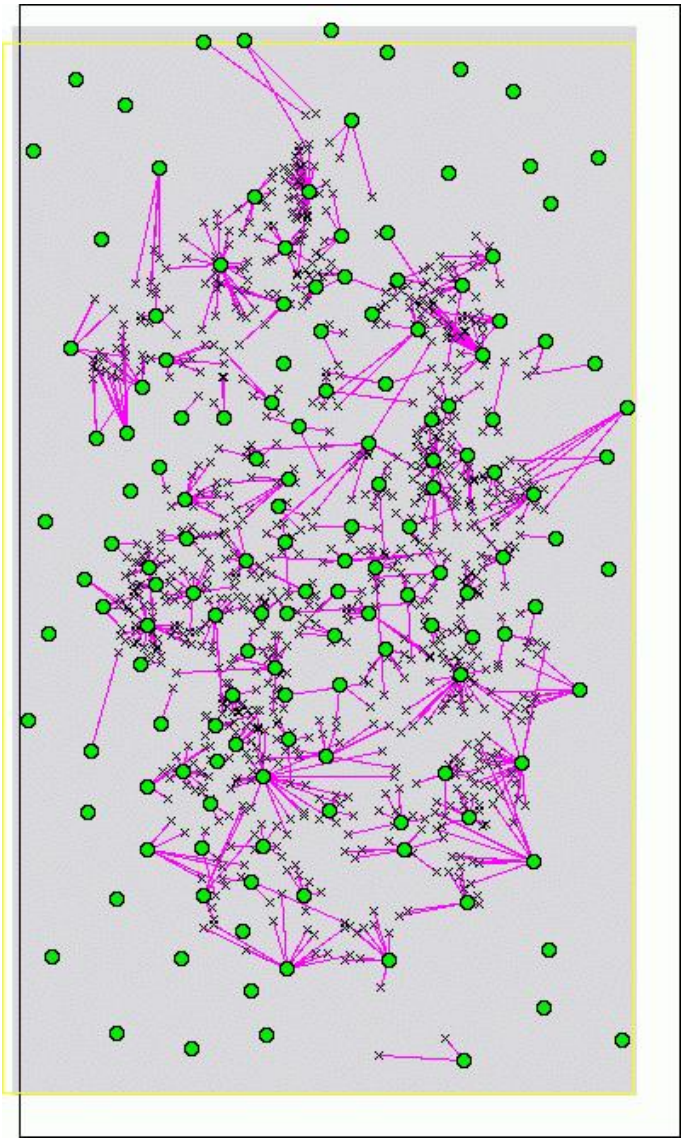
- use a binary bitstring, e.g., (101101)
- use a sequence of integer values in certain ranges, e.g., $(2, 6, 98, 3) \in [1:2] \times [1:10] \times [0:100] \times [1:5]$
- use a sequence of real values in certain ranges, e.g., $(1.23, 34.4, -2.1) \in [-50.0, 50.0]$
- use a permutation
- use a k-dimensional structure
- use a binary tree
- use a general graph
- use whatever you like (recall: do something, be happy...)

Remember: we need a bijection between genotype and phenotype and we need to implement crossovers and mutations that are able to explore the entire search space (or at least the region of interest).

GA: names taken from biology

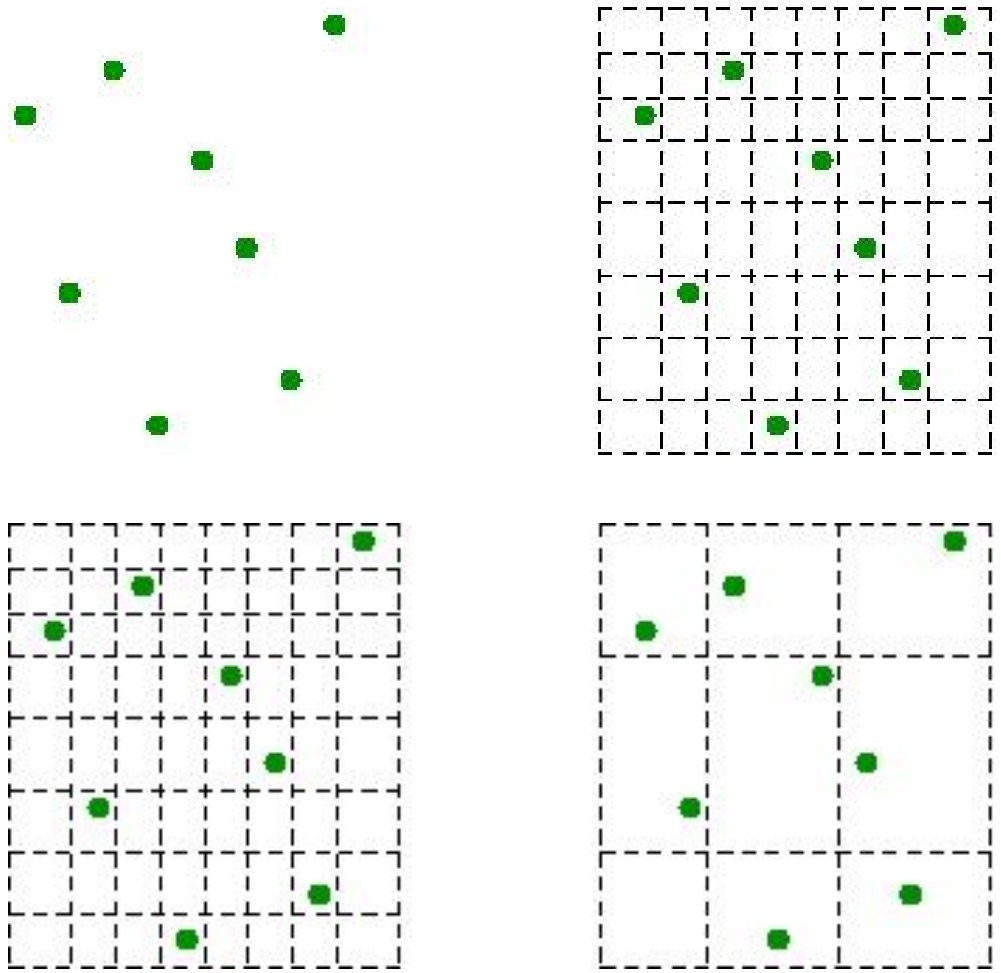
- The individual components of the sequences are called genes.
- The possible values of a gene are called allele.
- The encoding of an individual is called its genome or chromosome.

GA: genotype an example



- green: base stations
- crosses: mobile users
- magenta: assignment
- goal: find the minimal subset of base stations that guarantees an assignment of all mobiles
- Note: computation of the objective function is quite complex (and will not be detailed here).

GA: genotype an example



- initial
  8×8 grid
- reduced to
  3×3 grid
- 4 allele
  (2-bit strings)
  - unusable
  - used
  - unused
  - fixed

GA: mutation possiblities

gene mutation:
: just change randomly one (or more) gen(es) to another permitted allele

gene flip:
: interchange the values of two (or more) genes

gene sequence displacement:
: cut a sequence/part and insert at another position

gene sequence inversion:
: revert the order of a (partial) sequence

what-ever-you-like:
: do something, be happy...

GA: mutation rules of thumb

- The mutation rate should be, more or less, inversely proportional to the size of the genome.
- For larger populations maybe reduce mutation rate in the on-going optimization process.
- With low diversity in the populations, maybe a larger mutation rate helps.

GA: crossover possiblities

simple crossover:

parents	cut	children
(101101)	$(10\|1101)$	$\longrightarrow (100111)$
(010111)	$(01\|0111)$	$\longrightarrow (011101)$
$(2,8,98,3)$	$(2,8,\|98,3)$	$\longrightarrow (2,8,40,4)$
$(1,9,40,4)$	$(1,9,\|40,4)$	$\longrightarrow (1,9,98,3)$

k-point crossover:

cut at k points and interchange the corresponding parts (variation: take k at random)

uniform crossover:

interchange each gene with certain probability

multiple parent mating:

use more then two parents and interchange genes (variation: merge entire parent set)

GA: crossover possiblities (continued)

arithmetic crossover: assign to children convex combination of parent genes with some random weight, $\alpha \in [0,1]$, e.g., with $\alpha = 0.7$ on second gene:

$34.5 \cdot 0.7 + 13.5 \cdot 0.3 = 28.2$

$(1.23, 34.5, -2.1) \longrightarrow (1.23, 28.2, -2.1)$

$13.5 \cdot 0.7 + 34.5 \cdot 0.3 = 19.8$

$(10.5, 13.5, 23.1) \longrightarrow (10.5, 19.8, 23.1)$

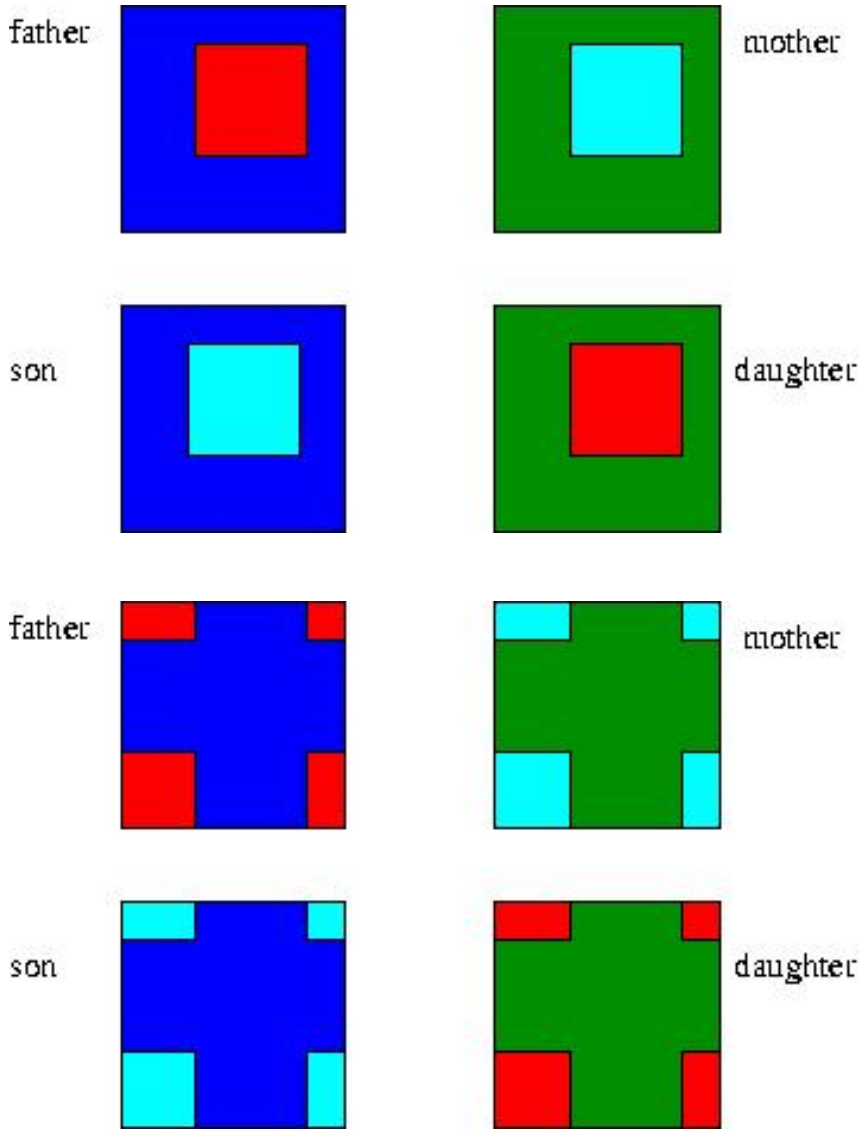
(again variations: as *k*-point, or with all genes, or with *k* at random)

blended crossover: blend two corresponding parent genes with a certain, usually fixed, value $\alpha \in [-0.5, \infty]$ according to the current gene spread

simulated binary crossover: blend two corresponding parent genes according to a suitable probability density function

what-ever-you like: remember, do something, be happy...

GA: crossover example (2-point cyclic crossover)



- select two grid points
- interchange rectangles

GA: principal loop

A genetic algorithm can be summarized in the following principal loop (parts that are dealt with marked with *DONE*):

```
InitializePopulation()        # initialization
EvaluateIndividuals()         # evaluation          DONE
while not Stopping():         # stopping
  DetermineParents()            # selection
  GenerateChildren()            # recombination     DONE
  MutateChildren()              # mutation          DONE
  EvaluateIndividuals()         # evaluation        DONE
  ReestablishPopulation()       # selection
```

GA: selection

- In the principal loop, there are two selection processes:
  - How to select the parents to generate the off-springs? and
  - How to rearrange the final population or next generation?
- We use the following notation:
  - μ stands for the number of individuals in the population
  - λ stands for the number of children being generated
- We distinguish two main strategies:
  - $(\mu + \lambda)$-strategy
    - from the μ individuals of the current generation select the parents and generate λ children
    - from the $\mu + \lambda$ individuals choose the μ best ones as new generation
  - (μ, λ)-strategy
    - from the μ individuals of the current generation select the parents and generate $\lambda \geq \mu$ children
    - from the λ children choose the μ best ones as new generation

The second question from above is answered.

GA: selection options

To select the parents being allowed to have off-springs, there exists a bunch of suggestions:

roulette wheel: assign to each individual a fraction of the wheel according to its relative fitness and spin the wheel (variation: smooth, weight, or normalize the fitness somehow, e.g., use log of objective function, or use z-score)

rank based: order the individuals according to fitness and select with a probability weighted by the rank (variation: compute selection probability with linear function of rank, so the least ranked still gets certain probability to get selected)

GA: selection options (continued)

tournament based: draw a certain number of random individuals, select the best one as parent (variation: select directly the best two as parents)

truncation selection: only the individuals with highest fitness values will be parents

what-ever-you like: remember, do something, be happy...

The first question (from two slides earlier) is answered.

GA: initialization

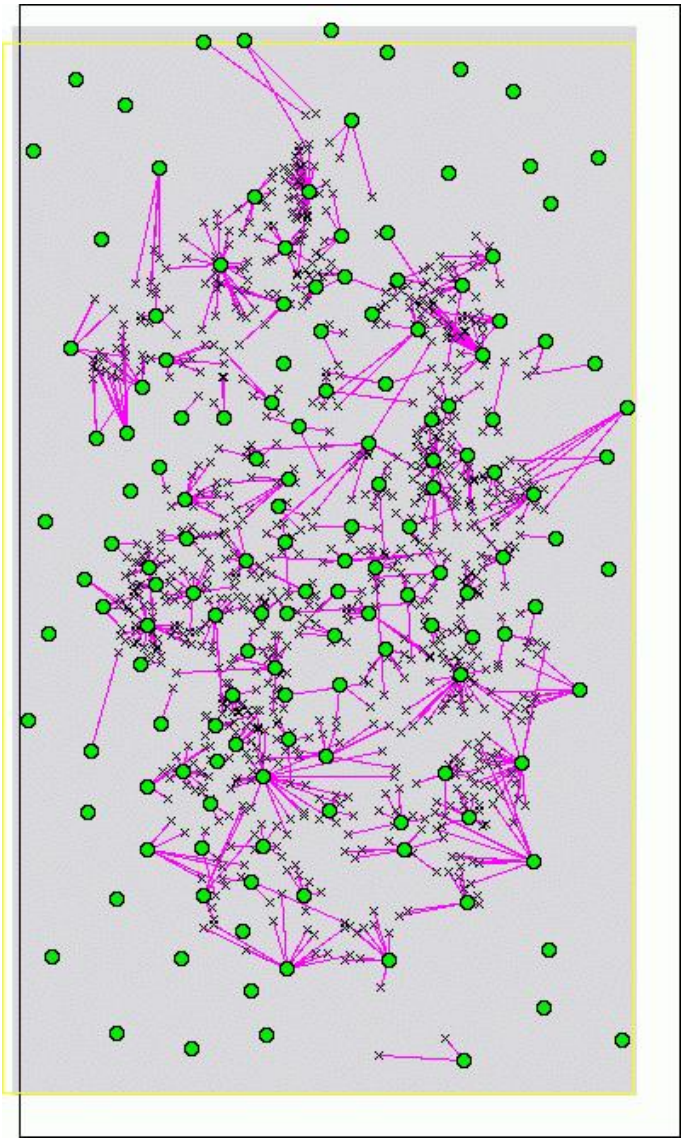
- generate the initial population with random genomes
  - take into account that the distribution according to genotype not necessarily is similar to the distribution in phenotype
  - maybe there are many individuals with very low fitness
  - the initial convergence rate might be slow
- generate the initial population with individuals from another heuristic algorithm or various such algorithms
  - the population might be biased into a certain region of the search space
  - the diversity of the population might be low
  - the convergence rate might be trapped early in a local optimum
- recommendation: use a mixture of both

GA: termination criteria

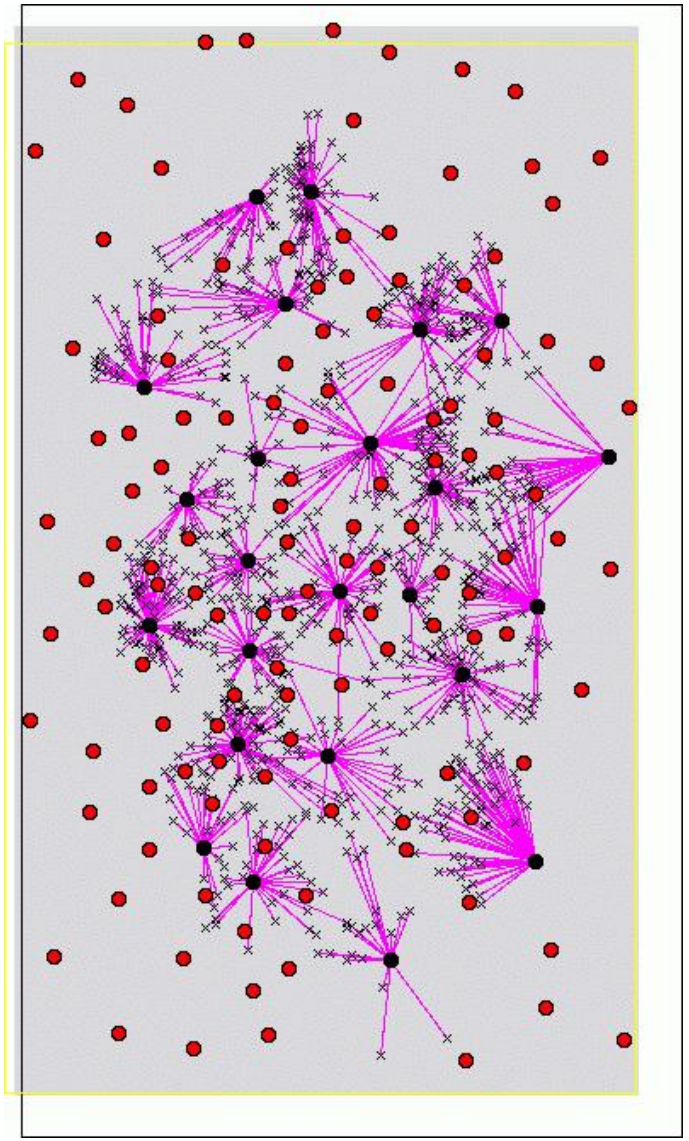
There are many possibilities when to stop the iteration of a genetic algorithm:

- once the first solution has been found
- once a sufficiently good solution has been found
- once the optimum has been found
- once a certain number of iterations has been executed
- once the diversity of the population is below a certain threshold
- once the convergence rate of the improvement is below a certain threshold
- once a certain amount of runtime has been spent
- once the iteration is stopped by user interruption
- recommendation: use an OR-mixture of all

GA: results for the example (2007)



119 of 149 nodes used

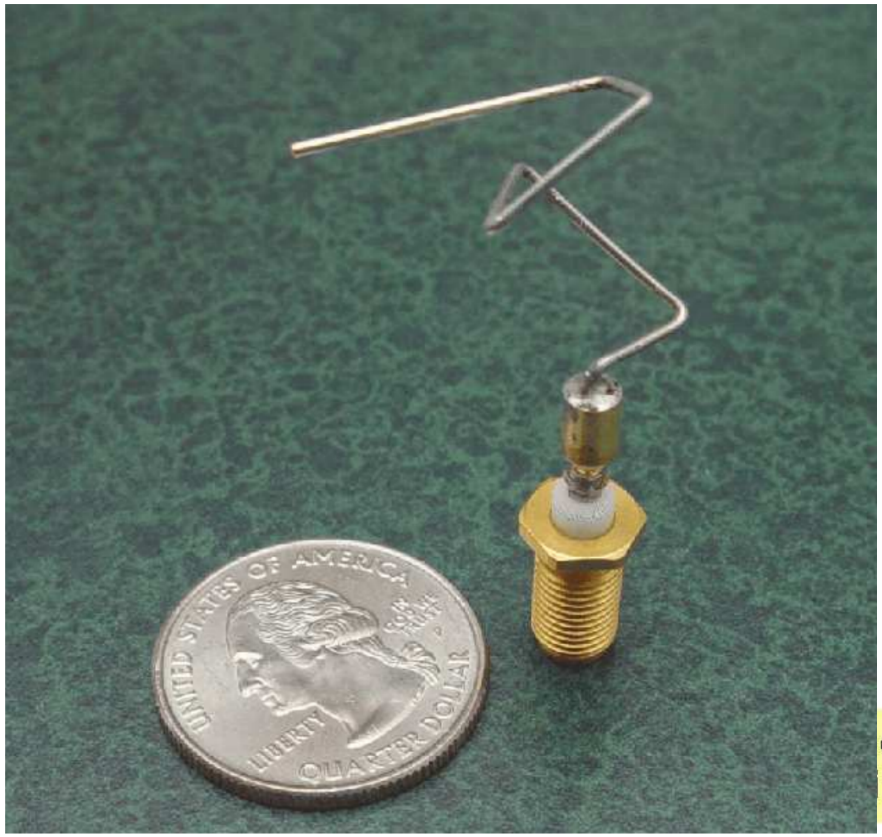


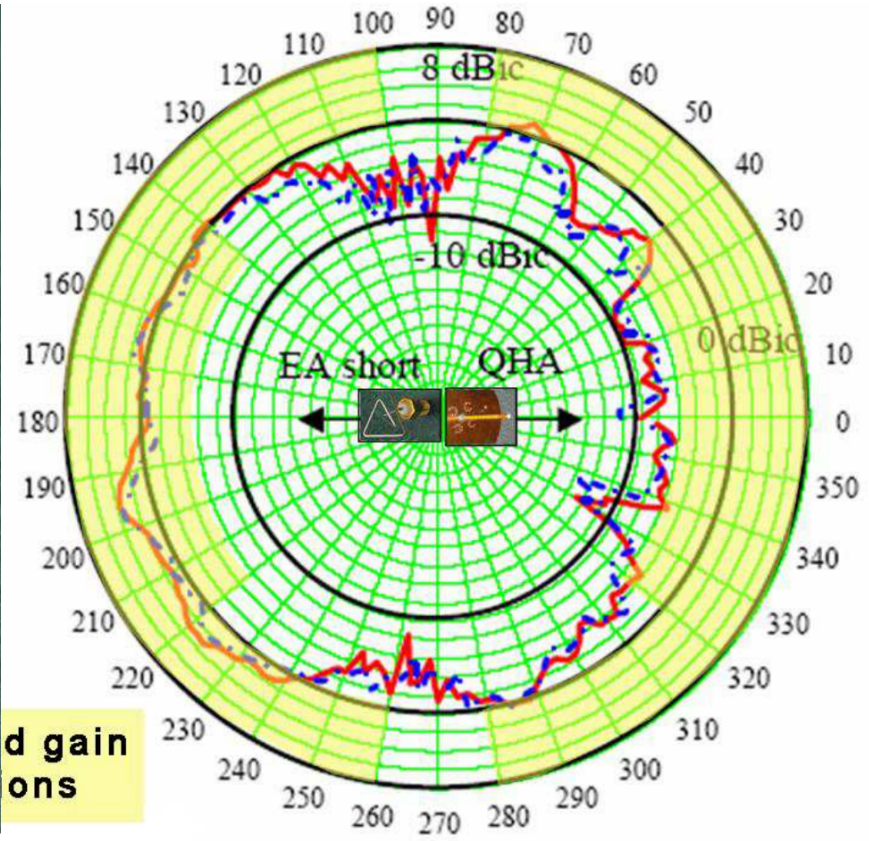
24 of 149 nodes used

GA: use for antenna design (small satellites)

Horny, AlGlobus, Linden, Lohn: Automated Antenna Design with Evolutionary Algorithms (2012)

https://arc.aiaa.org/doi/10.2514/6.2006-7242 or https://www.researchgate.net/publication/228909002_Automated_Antenna_Design_with_Evolutionary_Algorithms





d gain
ons