Unit 5: Genetic algorithms

A. Riscos Núñez
J. L. Ruiz Reina

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

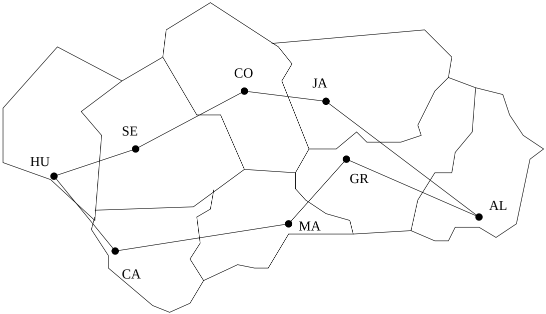
Introduction: optimization problems

- Algorithms for searching optimal solutions
  - Search for the **best** solution within the space of possible (candidate) solutions
  - Maximization or minimization
- Iterative improvement
  - Start on an "arbitrary" initial solutions
  - Improve it step by step
- Algorithms:
  - Hill-climbing
  - Random-restart hill-climbing
  - Simulated annealing
  - Genetic algorithms
  - ...
- None of the above algorithms is complete, but very often they are the only feasible option in practice.

Example: The travelling salesman problem

Problem:

Given a list of cities, find the **shortest** possible route that visits each city exactly once and returns to the origin city (assuming all of them are directly connected and their pairwise distances are known)



Genetic algorithms: Darwinian evolution

- Optimization inspired on the evolutionary process going on in Nature:
  - Evolution is carried out within the chromosomes of the individuals
  - The "good structures" get higher survival probabilities than the rest
  - New genetic material is obtained by crossover and mutations
- Genetic algorithms:
  - Application of such ideas for searching optimal solutions
  - There are many genetic algorithms
  - It is a general denomination for this kind of evolutive algorithms

Genetic algorithms: codification of the problem

- First step is to represent the states of the original problem as individuals of a population
  - **Genes**: basic genetic material
  - **Chromosomes**: sequence of genes codifying a state of the problem
  - **Population**: Set of chromosomes (not too large, "manageable")
  - The population evolves into different *generations*
  - Genotype vs. phenotype

Fitness of individuals

- According to the value of the **objective function**

- Optimization problems: a simple example
  - Example: find the minimum of thefunction $f(x) = x^2$ en $[0, 2^{10}) \cap N$
- Variables ***GENES*** and ***INDIVIDUALS-LENGTH***
  - In our example (square function): **`[0, 1]`** and **`10`**, resp.
- Function **`DECODE(X)`**, defines the *phenotype*
  - For the square function example: a chromosome can be read as a binary number of 10 digits (and in reverse order). The phenotype of a chromosome is such a number (in decimal notation)
  - Example: **`(0 1 1 0 0 1 0 0 0 0)`** is a chromosome representing number 38
- Function **`FITNESS(X)`**, is the value to be optimized (acting over the phenotype)
  - For the square function example: a function that gets a natural number and returns this number squared

Genetics algorithm: a general description

```
INITIALIZE population
EVALUATE each individual of the population

Repeat until HALTING-CONDITION
   SELECT parents
   COMBINE pairs of parents
   MUTE offsprings
   EVALUATE new individuals
   SELECT individuals for the next generation

Return the best individual of the last generation
```

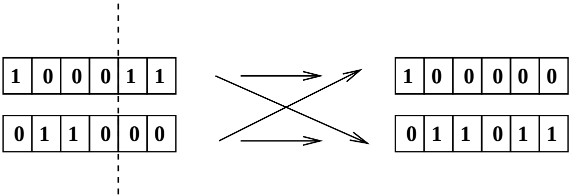
Example of execution (minimizing the square function)

```
>>> genetic_algorithm(sqr_gen, 20, 10, 0.75, 0.6, 0.1)

 Generation: 1. Average: 361954.9, Best: (0 1 1 0 0 1 0 0 0 0), Fitness: 1444
 Generation: 2. Average: 79730.6, Best: (0 1 1 0 0 1 0 0 0 0), Fitness: 1444
 Generation: 3. Average: 22278.6, Best: (0 1 1 0 0 1 0 0 0 0), Fitness: 1444
 Generation: 4. Average: 3537.7, Best: (1 1 1 1 0 0 0 0 0 0), Fitness: 225
 Generation: 5. Average: 1597.3, Best: (0 0 1 1 0 0 0 0 0 0), Fitness: 144
 Generation: 6. Average: 912.8, Best: (0 1 0 0 0 0 0 0 0 0), Fitness: 4
 Generation: 7. Average: 345.3, Best: (0 1 0 0 0 0 0 0 0 0), Fitness: 4
 Generation: 8. Average: 60.7, Best: (0 1 0 0 0 0 0 0 0 0), Fitness: 4
 Generation: 9. Average: 14.0, Best: (0 1 0 0 0 0 0 0 0 0), Fitness: 4
 Generation: 10. Average: 4.5, Best: (0 1 0 0 0 0 0 0 0 0), Fitness: 4
 Generation: 11. Average: 3.7, Best: (1 0 0 0 0 0 0 0 0 0), Fitness: 1
 Generation: 12. Average: 3.4, Best: (1 0 0 0 0 0 0 0 0 0), Fitness: 1
 Generation: 13. Average: 2.4, Best: (0 0 0 0 0 0 0 0 0 0), Fitness: 0
 ....
```

Combining individuals

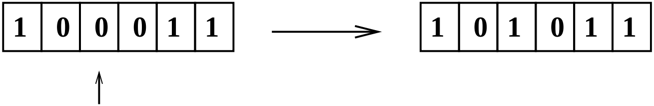
- Operators combining *parents* information to obtain new *offsprings*
- Single-point crossover:



  - Randomness: when chossing the crossover point
- Alternatives:
  - Multi-point crossover (several swapping points)
  - Uniform cross-over (for every offspring position, *randomly* choose from which parent "inherits")
  - Specific crossover, for particular representations (for example, permutations)

Mutations in individuals

- Mutations:



- Randomness:
  - With a given probability (usally low) change some genes
  - When changing, randomly choose the new gene
- Variants:
  - Specific of the representation (for example, permutations)

Selection mechanisms

- A genetic algorithm needs a method to select individuals from a population:
  - To select parents
  - Sometimes, to select which offsprings
- In general, the selection method must be biassed towards individuals having better fitness, but keeping some degree of *diversity*
  - Usually, with randomness
- Some popular selection mechanisms:
  - Fitness proportional
  - Tournement
  - Elitist + randomness

Fitness-proportional selection

- Idea:
  - Select randomly, but in such a way that each individual can be selected with a proablilty equal to the proportion of its fitness with respect to the total fitness of the population
  - Therefore, the better the individual, the more probable to be selected
- The probability of selecting individual i is

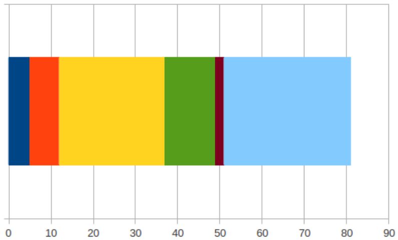
$$P(i) = \frac{F(i)}{\sum_{j=1}^{n} F(j)}$$

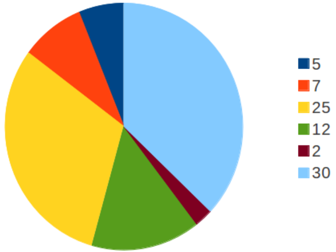
  - Important: this selection method can only be used when we have a maximization problem (why?)
  - In case of minimization, we can transform its *fitness*
- Variant: *ranking* selection
  - The probability to be selected is proportional to its relative position in the population (ordered by fitness)

Roulette-wheel selection method

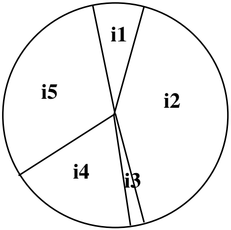
Selecting one chromosome

1. Calculate, for each individual in the population, its associated cummulative sum of the objective function values
2. Generate a random number x between 1 and the total sum of values
3. Iterate over the population, and return the first chromosome whose cummulative sum is greater or equal than x





Example of Roulette-wheel selection



- Populatin with 5 individuals, fitness 2,7,1,4,6 resp.
- Accumulated sums: 2,9,10,14,20
- For example, to select four individuals, we take four random values between 0 and 20: 7, 13, 17, 5
- Selected: individuals 2,4,5 and 2 (note that one individual can be seelcted more than once)

Tournement and elitism

- Tournement selection:
  - To select one individual, randomly choose k individuals and select the best
  - Advantages: it does not depend on the fitness magnitude, and it can be applied both to minimization and maximization
  - The larger the k used, the bigger the *selection pressure*
- Elitist selection:
  - Directly select some of the individuals among the best
  - To introduce diversity, randomly select individuals from the rest

Other components of a genetic algorithm

- Initial population:
  - Usually we create N individuals, randomly
- Termination of the algorithm:
  - A given number of generations
  - If fitness is not improved after a number of generations
  - Until we get an individual better than a given threshold fitness
- Parameters:
  - Population size
  - Parents proportion
  - Crossover and/or muaution probabilities

Genetic algorithm using tournement selection (pseudocode)

```
t := 0
Init-Population P(t)
Eval-Population P(t)

While not End do:
    P1     = Select by tournement (1-r)·p individuals from P(t)
    P2     = Select by tournement (r·p) individuals from P(t)
    P3     = Apply pairwise crossovers to P2
    P4     = Union of P1 and P3
  P(t+1) = Mutate P4
  Eval-Population P(t+1)
  t = t+1
End-While

Return the best individual in P(t)
```

- Parameters: population size (**p**), number of generations, parents proportion (**r**), mutation probability

A genetic algorithm with selection by elitism and randomness (pseudocode)

```
t := 0
Init-Population P(t)
Eval-Population P(t)

While not End do
   P'    := Select-Parents P(t)
   P''   := Crossover P'
   P'''  := Mutation P''
   Eval-Population P'''
   P(t+1)  := Select-Best P''',P(t)
   t:= t+1
End-While

Return the best individual in P(t)
```

- Parameters: population size, number of generations, proportion of parents, proportion of best individuals among parents, mutation probability

GA representation: example

GA representation for the Andalusian TSP

- **GENES** = (AL CA CO GR HU JA MA SE)
- **INDIVIDUALS-LENGTH** = 8
- **DECODE(X)= X**
- **F-OBJECTIVE(X)** = circuit length??
  - The codification allows repeated cities in chromosomes
  - A penalty is required for such individuals

GA representation: example

GA representation for the Andalusian TSP (cont.)

- Penalty for incomplete paths

  PENALTY(PATH)= 100 * |GENES - PATH|

- Objective function

  F-OBJECTIVE(X)= 2*DISTANCE-TRIP(X) + 50*PENALTY(X)

  - Combination of distance and penalty
  - The weights of each component can be adjusted experimentally

Genetic algorithm: practice

Experimental results for Andalusian TSP (illustrative):

Pop. size	Crossover %	From-best %	Mutation prob.
50	0.75	0.6	0.05

- One run:
  - Best individual found: (HU SE CO GR AL JA MA CA)
  - Value: 2015.8258
- After repeating 84 times:
  - Best individual found: (MA GR AL JA CO SE HU CA)
  - Value: 1859.8511 (optimal!)

Conclusion

- Genetic algorithms as a local search process
  - Iterative improvement
  - Crossover, mutations and diversity try to avoid the problem of local optima
- Many other implementations of genetic algorithms are available
  - Always based in the same principles: reproduction, mutation, biassed selection for the best, and looking after diversity
- Easily applied to many types of problems:
  - optimization, machine learning, planification,...
- Performance of the results acceptable in some problems
  - Although they do not compete against specific-purpose algorithms

Bibliography

- Russell, S. and Norvig, P. *Artificial Intelligence (A Modern Approach) 3rd edition* (Prentice–Hall, 2010).
    - Ch. 4 “Beyond classical search”.
- Mitchell, T.M. *Machine Learning* (McGraw-Hill, 1997)
    - Ch. 9: “Genetic Algorithms”
- Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs* (Springer, 1999).
    - Ch. 2 “GAs: How Do They Work?”.