Evolutionary Computation

2024/25

Master Artificial Intelligence

Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

24/25

multi-objective evolutionary algorithms (MOEA)

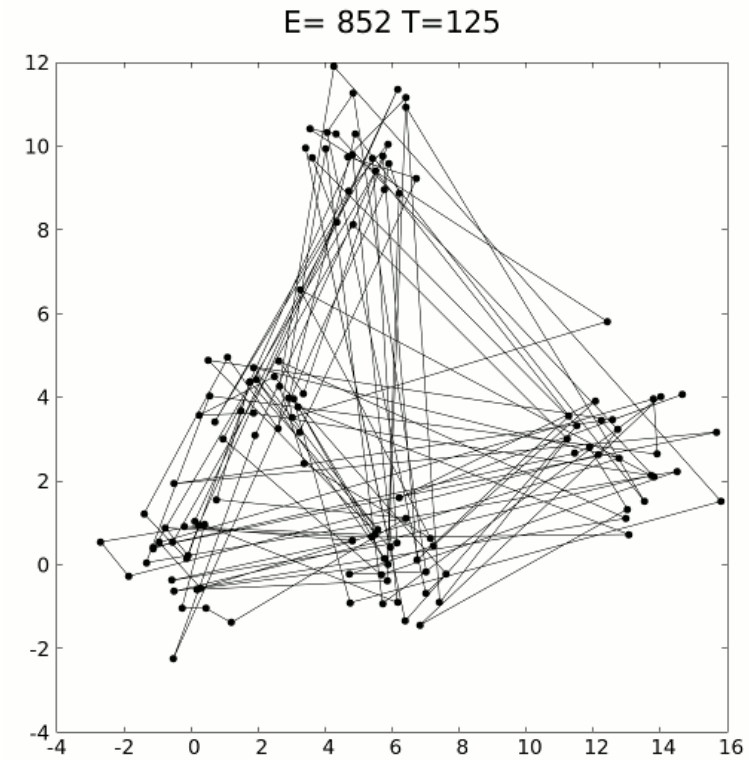
- VEGA, vector evaluated genetic algorithm
  Idea: in the selection process parts of the mating parents are selected according to each objective function
- MSGA, multi-sexual genetic algorithm
  Idea: individuals are marked as belonging to a certain objective function, ranking is used to select parents, only differently marked individuals are used to generate children
- NSGA, non-dominant sorting genetic algorithm
  Idea: sort individuals according to their dominance, and design the selection according the dominance classes (i.e., work with several fronts, intending to converge eventually to the Pareto front.

multi-objective evolutionary algorithms (MOEA)

- NSGA-II, including elitism, dominant individuals are preserved in population, clustering is avoided
- SPEA, Strength Pareto Evolutionary algorithm
  Idea: maintain a fixed set of best individuals while guaranteeing that they are spread over the Pareto front without to much clustering

Simulated Annealing

The name and idea stem from the slow and repeated heating and cooling process of certain materials (usually alloys of different metals with addings, e.g., iron+carbon) until certain properties are achieved.



https://en.wikipedia.org/wiki/Simulated_annealing

Simulated Annealing

- Explores a neighbor in the search space, whenever
  - there is an improvement in the objective function at the neighbor or
  - it is not worse than a temperature dependent threshold with some probability.
- Let Δf be the difference between current solution f and neighbor solution f_n.
- Let T be a temperature.
- Then the neighbor is accepted whenever either $\Delta f < 0$ (we are going downhill)
  or if $e^{-\Delta f/T} > r$, for r being a random value in $[0 : 1]$
  (we are going uphill).
- With increasing iteration rounds (Monte Carlo iterations where the temperature is held constant) the temperature is reduced, hence, the threshold converges towards zero.

Simulated Annealing cooling schemes

- linear cooling: $T = T_0 - \eta k$
  with T_0 an initial temperature, k the iteration number and η some free parameter, being constant during optimization.
  (To avoid negative temperature: $T = \max(T_0 - \eta k, T_{\min})$)
- exponential cooling: $T = aT$
  with $a \in [0.8, 1)$ typically; slower cooling the a close to 1.
- inverse cooling: $T = T/(1+\beta T)$
  with β being a small constant (e.g. $\beta = 0.001$).
- logarithmic cooling: $T = T_0/\log k$
  with c a suitable constant.
  (Used as well a generalization: $T = T_0/\log(k+d)$).
  Not really practical in applications but was used to prove convergence to global minimum under certain conditions.
- inverse linear cooling: $T = T_0/k$.
  Not really practical in applications but was used to prove convergence to global minimum under certain conditions.

Simulated Annealing cooling schemes

- It seems that any monotonely decreasing function, that is neither too steep at the beginning and neither too slow decreasing in the end, might serve.
- Cooling can be implemented differently in each dimension in multi-dimensional problems.
- The entire process can be restarted from another location in search space (Monte Carlo approach around inner simulated annealing process).

Simulated Annealing vs. Tabu Search

- Both methods belong to the random path methods:
  - there is only one individual
  - the moves in the search space explore the neighborhood in a random manner
  - a move is accepted whenever
    - TS: a random, but not tabu, move among the improving once in first place and non-improving onces in second place.
    - SA: a random move that fulfills the annealing condition
- Tabu search is, as improvements are prefered, a local search method that finds local minima;
  which might not be the case for simulated annealing (you need to add some final local search approach).

memetic algorithms

The population based algorithms usually do not perform local optimization of the individuals.

Including such an approach is called a memetic algorithm:

- use an adequate local search strategy (e.g. steepest decent, iterated local search etc.) to improve the fitness of the individuals
- this improvement of an individual is also called *individual learning*
- the result of the local optimization:
  - might change the genotype of the individual (Lamarckian learning), i.e., the changed individual participates in the genetic algorithm,
  - or might not change the genotype of the individual (Baldwinian learning), i.e., the population is not changed

memetic algorithms efficiency considerations

- the local search can be restricted to a certain part of the population, for instance, the best ranked ones
- the local search can be restricted to be performed only after a certain amount of iterations in the overall genetic algorithm

biogeography-based optimization

- The biogeography-based optimization uses the idea of populations evolving at islands that once in a while exchange individuals via migration.
- Implemented as further meta-heuristic in the framework of genetic algorithms (especially for separable objective function, where optimization of their convex combination is an option).
- Can be applied to particle swarm optimization as well (I have found implementation examples, e.g. from 2017).

much more nature-inspired work done

Without going into details, there is a huge bunch of other nature-inpired optimization heuristics (see introduction as well):

- ant lion optimization
- artificial bee colonies
- bat algorithm
- dragonfly algorithm
- firefly optimization
- grasshopper optimization
- grey wolf optimizer
- whale optimization
- invasive weed optimization
- cristalization of materials
- great deluge algorithm
- gravitational search algorithm
- etc. etc. etc.

issues to be considered

The following issues should be considered when implementing and using a certain optimization approach for a given problem:

- implementation difficulty
- number of parameters to be adjusted
- population size
- number of objective function evaluation
- convergence velocity (convergence profile)
- handling of local optima (stucking)
- handling of restrictions
- statistically correct evaluation on benchmark problems and Monte Carlo runs
- is there a Las Vegas condition available?

efficiency issues to be considered

The following efficiency aspect might be useful in a certain implementation of an optimization approach on a given platform:

- use of parallelization
- use of caching
- use of efficient data structures
- use of adequate precision in calculation
- use of approximation algorithms (especially in early phases for the objective function computation)
- use good random number generators

hyper-parameter tuning

Many of the different heuristics seen so far have a certain number of free parameters that must be fixed when starting the optimization, but

- we might use another optimization algorithm that tries to find a good parameter set for the original algorithm
- for instance, we run the algorithm on a suitable benchmark suite to obtain good free parameters, and then run the optimization with these settings on the real problems we are interested to solve
- this process can be iterated, i.e., each use of the optimizer might adapt to a certain degree its free parameters
- eventually we arrive at an online-algorithm which adapts its parameters with each problem it is faced to

last but not least

that’s it, folks ... thanks. questions?