Evolutionary Computation

2022/23

Master Artificial Intelligence

Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

22/23

particle swarm optimization (PSO)

- The inspiration comes from social behavior of individuals within an environment including other individuals.
- We work with n individuals that move in a continuous d-dimensional search space.
- The individuals move (in steps) through the search space and adjust their velocities according to information gathered from others (and their own *histories*).
- The individuals are grouped into neighborhoods.

PSO: velocity actualization

- x_i vector of current positions
- v_i vector of current directional velocities
- b_i best local position vector
- h_i best neighbor position vector
- $\varphi_1 = 2.05, \varphi_2 = 2.05$ influence values (just some *magic*)
- $\xi \in [0.4, 1]$, e.g. $\xi = 0.729$ inertia reduction value
- velocity actualization

$$\begin{aligned} v_i &= \xi v_i + U[0, \varphi_1] \circ (b_i - x_i) + U[0, \varphi_2] \circ (h_i - x_i) \\ x_i &= x_i + v_i \end{aligned}$$

- The $\circ$ operator is either a Hadamard-operation (i.e., component-wise), or a linear operation (i.e., scalar multiplication)

PSO: principal loop

A particle swarm optimization can be summarized in the following principal loop:

```
InitializePopulation() # i.e. x_i, v_i
EvaluateIndividuals()  # i.e. b_i
DefineNeighborhoodSize()
while not Stopping():
  DetermineNeighborhoodValues()   # h_i
  UpdateIndividuals()             # i.e., x_i, v_i, b_i
```

PSO: some more details

- The velocity can be confined not to pass a certain maximum velocity. This feature helps to avoid explosion, i.e., that the area of the search space explored by the particles becomes larger and larger exponentially.
- Initial velocities can be zero or some random values.
- Small neighborhoods tend to provide a better global search, while larger neighborhoods tend to produce faster a convergence (but maybe premature).
- Neighborhoods can be defined as nearest neighbors, as fixed and overlapping, or entail the entire population, or what-ever-you-like.
- The inertia reduction can be increased with the simulation time.
- The best global individual g can be included in the equation: add $+U[0,\varphi_3] \circ (g - x_i)$
- The worst (local and globals) positions can be included to be *avoided*: add $-U[0,\varphi_4] \circ (\overline{b}_i - x_i)$ and/or $-U[0,\varphi_5] \circ (\overline{h}_i - x_i)$ and/or $-U[0,\varphi_6] \circ (\overline{g} - x_i)$

PSO: different versions

binary version: the variables are interpreted as binary values according to a distribution or threshold

discret version: the variables are interpreted as integer values (for instance with simple rounding)

dynamic version: the search space is reinitialized and/or the local variables are reset (type of outer Monte Carlo loop)

PSO: convergence

- the individuals should exhibit certain diversity (recall the similarity measures)
- diversity can be forced dynamically by adapting the parameters alongside the simulation time
- or one might use the lack of diversity as a stopping condition

ant colony optimization (ACO)

The idea stems from stigmergy: exercise indirect communication and coordination through the environment (leave a trace and act on findings).

- The individuals of a population leave information (pheromones) in the search space.
- The decisions are based on individual information or behavior and on the pheromones encountered.
- The information (pheromones) is volatile and can evaporate.
- The pheromones or a statistical evaluation of the individuals define the solution.
- The inspiration stems from ants, bees, termites, wasps, etc.
- Initially invented to deal with combinatorial problems (like TSP).

ACO: principal loop

An ant colony optimization can be summarized in the following principal loop:

```
InitializePheromoneValues()
while not Stopping():
  for individuals in range(n):
    ConstructSolution(individual)
    UpdatePhermoneValues()
    UpdateIndividuals()
```

ACO: how TSP can be approached

- The ant colony optimization takes place on the graph of the underlying problem (here think of the complete graph among all cities).
- The ants are placed at the cities.
- The initial pheromones are placed on the edges (either constant value or inversely proportional to the distance).
- The ants (in an appropriate iteration) run along a path in the graph (excluing already visited cities) and draw at each city a decision in which direction to continue.
- The decision is based on pheromones on each possible edge, maybe on some own information stored at the individual, and on a random value.
- Once the tour is completed for all ants, all of them deposit their pheromone on their tracks.
- The general evaporation process is applied to all (changed) edges.
- The currently best tour is memorized.

ACO: when to use?

ACO approaches are especially possible when the underlying problem allows for a constructive solution (as seen with the nearest-neighbor heuristic for the TSP). Simon gives the example that an ACO approach found a tour with 3 % deficit on the Berlin52 problem.

The no-free-lunch theorem

The no-free-lunch theorem states that the performance of all optimization (search) algorithms, amortized over the set of all possible functions, is equivalent. The implications of this theorem are far reaching, since it implies that no general algorithm can be designed so that it will be superior to a linear enumeration of the search space (exhaustive search).

What are practical implications of the no-free-lunch theorem?

- Each problem (or each type/class of problem) might need its own and proper optimization method.
- Maybe for interesting problems we find good optimization algorithms (we are not interested in *all* problems).
- Benchmarking optimization algorithms is a challenge, as general benchmarks might just provide *average* data, but our algorithm might be special for a niche of problems.
- There is a need to categorize problems and algorithms to obtain some insight on which type of problem a certain type of algorithm performs well.

How to compare different approaches?

In order to compare different algorithms one might take into account:

- wall clock runtime on comparable systems
- (average) number of objective functions evaluations
  (but the rest of the inverted time must not be neglected)
  difficult to be used when comparing constructing algorithms
- the result as distance to optimium or to some known lower bound
- mean best fitness
- properties of the solution histogram (fitness of all solutions found)
- scaling properties with problem size (applied to any measure above)

Practical aspects to be considered

One has to decide what is really needed:

- need a good (or best) solution independent of runtime (e.g. controler for space telescope or the evolved antenna)
- need a moderate solution fast (e.g., daily TSP with time windows, where finding a feasible solution is already NP-hard)