Darwinismus:

- pouze ti silnější jedinci přežijou
- evoluční se tak udrží pouze ty výhodné informace
- trvá dlouho dílo se vyvinout

Lamarchismus

- získané vlastnosti je možno dědit
- rychlejší vývoj
- ve skutečnosti to tak nefunguje

Altruismus

- jednání zvyšující šanci přežití ostatních druhů
- čistý altruismus by byl evolučně k ničemu
  - otec altruista ale může tak chránit geny svých potomků

Baldwinismus:

- those quickly adapting to changes are the ones winning evolution selection

Co je EA:

- populační stochastický výpl. alg.
- robustní metaalgoritmus

SGA:

Selection → Crossover → Mutation

Roulette-wheel-selection: $p_i = \frac{f_i}{\sum f_i}$

Tournament → fitness compared with only limited items.

SUS → kolo s uniformě rozdělenými segmenty $1/n$

2-arm bandit:

- exponentially more trials to the winning arm
- but still playing the worse arm as well

C → SGA does exactly the same

Schemata:

$$m(S, t+1) \geq m(S, t) \cdot N \cdot \frac{f(S)}{\sum_i f(i)} \cdot \left(1 - p_c \frac{d(S)}{l-1} - p_m \cdot o(S)\right)$$

pro velké $f(S)$, malé $d(S)$ a $o(S)$ roste exponenciálně růst

above-average, short, low-order will survive

Their problems:

- collateral convergence → silné schéma 111**... roste v populaci vygeneruje hodně takových prefixů, takže schema ***000... je pak dost blízko
- operations are not independent actually
- population is too small and finite → the exponential growth is harmed by errors etc.

The good:

- GA implicitně vyhodnocují velké množství schémat
- zdůvodňují, že GA fungují na principu „building blocks"

$\delta(S)$ = délka mezi první a poslední fixní pozicí

$o(S)$ = počet fixních bloků

Integer encoding

- variable selected uniformly from domain / modified from previous
- in some cases (ordering) random mutation does not make sense

Evolutionary strategies:

M – počet individuí

L – počet nových individuí

R – počet „rodičů"

(M+L) – vybírám M z M+L

(M,L) – vybírám M z L (rodiče vždy zabíjím)

– mutace je způsob pohybu v prostoru $f(x)$

$y(t) = x(t) + N(0,s)$

pokud vylepšuje fitness, nahrazuji, jinak nechávám

p : 1/5 → exploit vs. explore

$p \geq 1/5$ $s = s \cdot c$ → explore more

$p < 1/5$ $s = s/c$ → exploit more

– endogenní parametry individua pro řízení mutace

– crossover → gangbang (takes average from R for example)

Differential evolution:

$\forall v_{i,p}$: vyberu $v_{a,p}; v_{b,p}; v_{c,p}$

– donor: $v_{a,p} + F \cdot (v_{b,p} - v_{c,p})$

– pak crossover je přes každou pozici s donorem

– selekce je maximalizující výběr ← vždy alespoň jeden z donoru
mezi původním a zcrossovaným prvkem

– mutace typu random/1, best/1 apod.

Particle swarm optimalization

– each particle has memory of its movement

update looks like:

$v \leftarrow v +$

$c_1 \cdot rand(0,1) \cdot (pBest - present) +$ → towards local optimum

$c_2 \cdot rand(0,1) \cdot (gBest - present)$ → towards global optimum

Evolutionary ML:

Michigan

- individual is a single rule

|1|0|1|*|1| ——> action

- each rule has weights — if more applicable, stronger wins
- population is an expert
- evolve only sometimes

LCS ~ bucket brigade
- rules are applied only when they pay, the success is distributed by the paid amount.
- messages & receptors

ZCS
- matched rules selected with policy (roulette ...)
- winning rules are awarded with $1/|A|$ portion of reward, so are their predecessors — A ~ set of actions applicable
- has very short rewarding history

cons: - rule is not good by how many times I can use it

alg:
- $(b \cdot r)/|A|$ for winners
- $(g \cdot B)/|A_{-1}|$ for predecessors → B zůstává snižování všech strenghts [illegible]

XCS

pros: - rule is good by how well it affects the result...
- rule is (c, a, p, e, f): condition, action, payoff precondition, prediction error, evaluation function
- generate match set, group by actions, get avg. fitness
- take the max (or stochastic) action
- redistribute payoffs to the rules in the previous action set
- works on payoff predictions
- Q-learning alg.

Pitsburg

- individual is a set of rules
  - rule priorities, conflicts
  - more complicated operators

GIL:
- simple classification — no right-hand side of a rule.
- individual is a DNF formula (disjunction of complexes
- operators on ind. level:
  - swap of rules / copy / generalization / deletion
- operators on complex level
  - split of complex / generalization / specialisation
- operators on selectors
  - mutation / extension / reduction

Multiobjective optimization

- we have vector of fitnesses for each individual
- Scalarization → weighted sum and then SOA
- ε-constraint scalarization → one f_i chosen, other constrained

VEGA:
- sort population by each f_i, select some, cross, mutate and give back.
- tends to local optima of each single obj. func.

NSGA:
- population divided into fronts F_1...
  - F_1 - all non-dominant from P
  - F_2 - all non-dominant from $P - F_1$
  - ⋮
- each ind. has „niching" factor, by which the fitness is scaled before distribution: $1-(d(i,j)/d_s)^2$ ($d(i,j)$ – dist from i to j; d_s – param)
- lower front gets lower fitness

Combinatorial optimization:

Knapsack:

encoding: bitmap

operators: crossover, mutation, selection

→ all simplest

fitness: $\max\left(\sum_i v_i\right) \wedge \min\left(c_{max} - \sum_i c_i\right)$

TSP:

permutation repr:

- representation cities permutation

jak vypadají crossovers:

PMX

- preserves the most cities on their orig. pos.

(123|4567|89) PMX (452|1876|93)

1-4, 8-5, 7-6, 6-7

(423|1876|59) (182|4567|93)

OX

- relative order of cities is preserved

(123|4567|89) OX (452|1876|93)

9-3-4-5-2-1-8-7-6

8-9-4-2-3-5-6-7

(218|1876|93) (345|4567|92)

ER

- preserving as many edges as possible
- make list of edges for each city,
always take the city with least edges

mutace:

- subpaths shifts
- 2 cities swap
- subpaths swap

SAT:

encoding:

bitmap – simple, but difficult

floating point: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \bar{x}_4)$

$f(y) = (y_1-1)^2 \cdot (y_2+1)^2 \cdot (y_3-1)^2 + (y_1+1)^2 \cdot (y_3-1)^2 \cdot (y_4+1)^2$

– minimizing the formula

Scheduling (school)

- direct matrix encoding
  - row is teacher, column is class, values are subjects.
- mutation = mixing subjects
- crossover = swap better rows from individuals

Genetic programming

Tree-based programming

- terminals / non-terminals
- crossover = subtree exchange
- mutation = subtree replacement
- fitness = by running the problem

Linear GP

- simple byte code
- easy operators, many meaningful programmes
- faster evaluation

Graph GP

- programme is DAG
- considering extensions of a programme
- complicated operators