EVA I

NAIL 025 – 2016/17

Roman Neruda

ENGLISH VERSION – 13-01-2017

Topics, sources, outlines.

INTRODUCTION

Literature

- Mitchell, M.: *Introduction to Genetic Algorithms*. MIT Press, 1996.
- Eiben, A.E and Smith, J.E.: *Introduction to Evolutionary Computing*, Springer, 2007.
- Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs* (3ed), Springer, 1996
- Holland, J.: *Adaptation in Natural and Artificial Systems*, MIT Press, 1992 (2nd ed).
- Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

Topics

- Evolution models, population, recombination.
- Genetic algorithms. encoding, operators, selection, crossover, mutation.
- Natural selection, simulation, objective function, roulette wheel, tournament, elitism.
- Representational schemata, schemata theorem, building blocks hypothesis.
- Prisonner‘s dilemma, strategies, equilibria, evolutionary stability.
- Evolution strategies, cooperation, meta-parameters.
- Differential evolution, CMA-ES.
- EA and combinatorial problems, NP-hard tasks, TSP, ...
- Machine learning and data mining, evolution of rule-based systems, Michigan vs. Pittsburgh.
- Learning classifier systems, bucket brigade algorithm, Q-learning.

Biological motivation, basic parts

EVOLUTIONARY ALGORITHMS

Darwin evolution theory

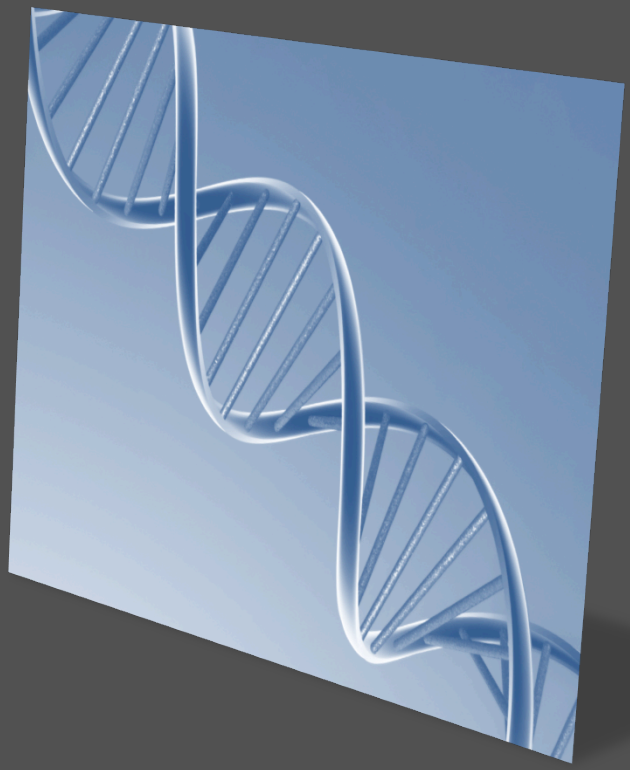
- 1859 – On the origin of species
- Limited environment resources
- Reproduction is the key to life
- Better fitted (adapted) individuals have bigger chances to reproduce
- Successful phenotype traits are reproduced, modified, recombined

Mendel genetics

- 1856 - Versuche über Pflanzenhybriden
- Gene as a basic hereditary unit
- Every diploid individual has two pairs of allels, one is transmitted to offspring independently of others.
- It‘s complicated:
  - Polygeny – more genes influence one trait
  - Pleiotropy – one gene influences more traits
  - Mitochondrial DNA
  - Epigenetics

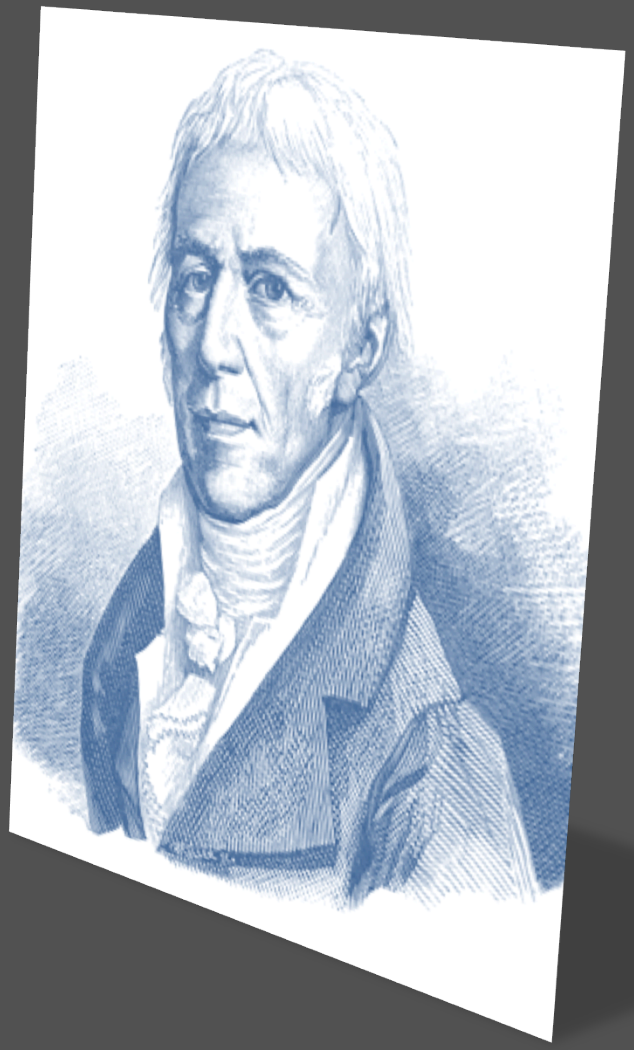
DNA

- 1953 – Watson&Crick – double helix structure of DNA
- Molecular-biological view:
  - How is the genetic information stored in a living organism
  - How is it inheritted
- DNA consists of 4 nucleotides/ bases – adenin, guanin, cytosin, thymin
- Codon – a tripplet of nucleotides encoding 1 out of 23 aminoacids (redundancy)
- These 23 aminoacids are the basic building structure of carbohydrates in all living organisms



Molecular genetics

- Crossover
- Mutation
- Transcription: DNA->RNA
- Translation: RNA->protein
- GENOTYPE->PHENOTYPE
- One-direction, complex mapping
- Lamarckism:
  - There is an inverse mapping from phenotype to genotype
  - Acquired traits can be inherited

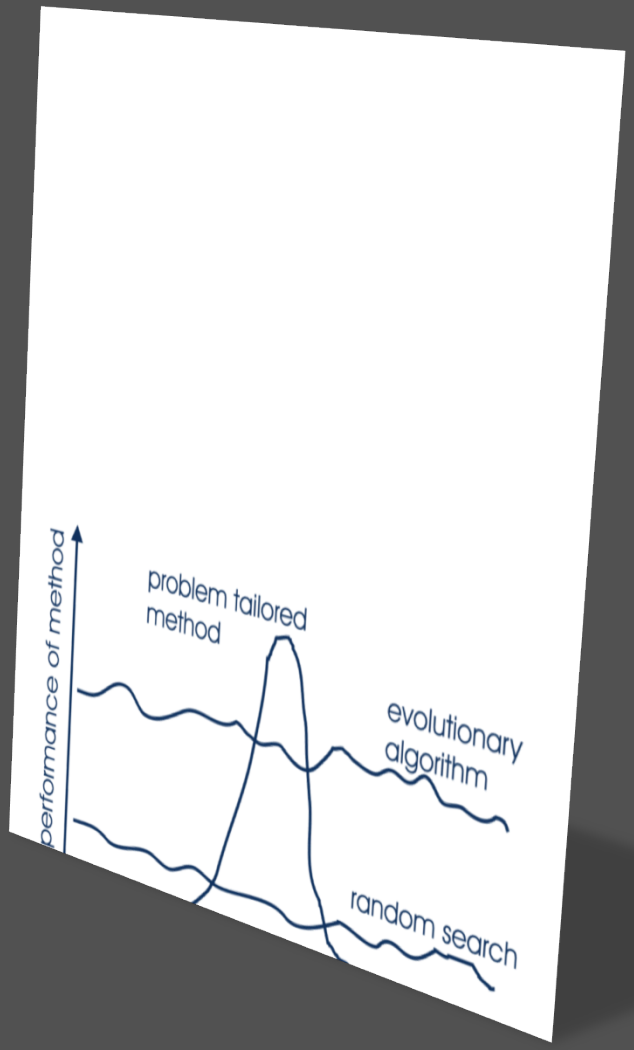


EA - summary

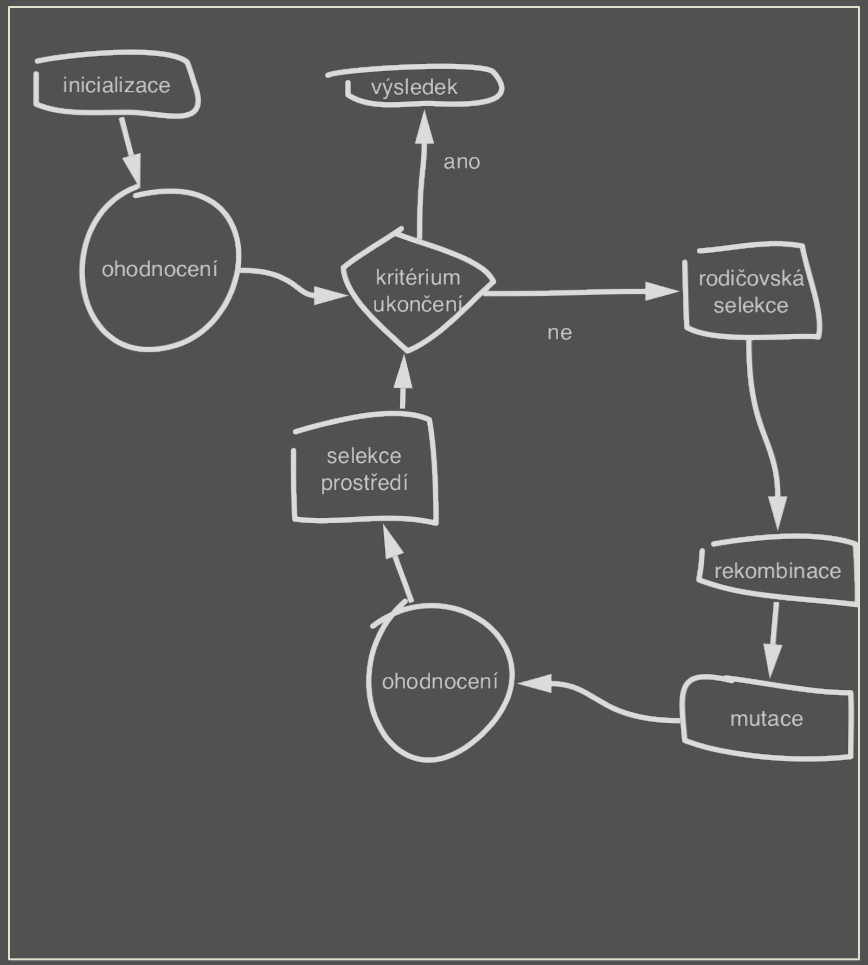
- Natural evolution: environment, individuals, fitness
- Artificial evolution: problem, candidate solutions, quality of a solution measure
- Eas are population-based stochastic search algorithms
- Recombination and mutation create variability
- Selektion leads the search in the right direction

General EA

- EAs are robust meta-algorithms
- No free lunch theorem – there is no one best algorithm
- It pays to create domain-specific variants of EAs
  - Representation
  - Operators



General EA



- Create initial population P(0) at random
- In a cycle create P(t+1) from P(t):
  - Parental selection
  - Recombination, and mutation
  - New individuals P‘(t+1) are created
  - Environmental selection chooses P(t+1) based on P(t) a P‘(t+1)

Genetic algorithms

- 1975 - Holland
- Binary encoded individuals
- Roulette-wheel selection
- 1-point crossover
- Bitwise mutations
- Inversion
- Schamata theory to explain the mechanism how GAs work

Evolutionary programming

- 1965 – Fogel, Owens a Walsh
- Evolution of finite automata
- No distinction between genotype and phenotype
- Focus on mutations
- No crossover, usually
- Tournament selection

Evolutionaty strategies

- 1964 - Rechenberg, Schwefel
- Optimization of real number vectors in difficult computational math problems
- Floating point encoding of individuals
- Mutation is the basic operator
- The mutation step is heuristically controlled or udergoes an adaptation (evolving)
- Deterministic environmental selection

Genetic programming

- 1992 – Koza
- Evolution of individuals representing (LISP) trees
- Used (not only) to evolve computer programs
- Specific operators of crossover, mutation, initialization
- Further applications (neuroevolution, evolving hw, …)

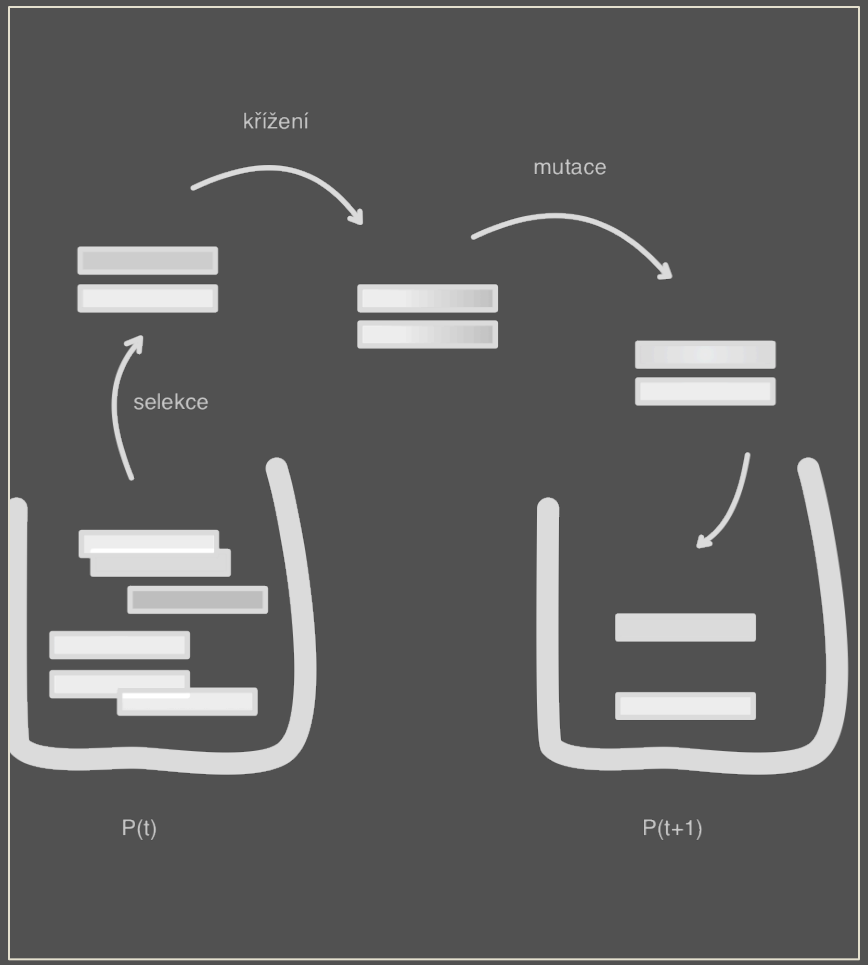
Holland SGA, binary reprezentation, operators and their variants

SIMPLE GENETIC ALGORITHM

GA

- Genetic algorithms – 70s USA, Holland, DeJong, Goldberg, …
- The original proposal is nowadays called SGA (simple GA)
  - Minimal set of operators, the simplest individual encoding, research of theoretical properties
- Gradually, the SGA has been enriched of – or transformed to – further operators, encodings, ways of dealing with populations, etc.

SGA - basics



- t=0; Generate at random initial population P(0) of n l-bit genes (individuals)
- Step from P(t) to P(t+1):
  - Compute f(x) for each x from P(t)
  - Repeat n/2 times:
    - Select a pair x, y from P(t)
    - Cross over x, y with probability p_C
    - Mutate every bit of x and y with probability p_M
    - Insert x, y to P(t+1)

Selection

- *Roulette wheel selection:*
  - Selection mechanism is based on the individual fitness value
  - Expected number of individual selections očekávaný should be proportional on the ratio of its fitness and an average fitness of the population
  - Roulette wheel selection: each individual has an allocated slice of a roulette wheel corresponding to its fitness, the wheel is spun n-times

Křížení

- V GA je křížení hlavním operátorem
- Rekombinuje vlastnosti rodičů
- Doufáme, že rekombinace povede k lepší fitness
- *Jednobodové křížení:*
  - náhodně zvolíme bod křížení,
  - vyměníme odpovídající části jedinců
  - Pravděpodobnost pC typicky v rozsahu desetin

Mutation

- In simple GA, mutation operator is less important, acts as a mechanism against stuck in local extrema
- (On the contrary, in EP nebo early ES, mutation is the only source of variability)
- Bit-string mutation:
  - With probability $p_{M,}$ every bit of the individual is changed
  - p_M is small (eg. to change 1 bit in individual on average)

Inversion and other

- The original Holland's SGA proposal contains another genetic operator – *inversion*
- *Inversion*
  - Reversing a part of the bit string
  - BUT with keeping the meaning of bits
  - More complicated technically
  - Inspiration in nature
  - Did not proven to be beneficial

Schema theorem, building blocks hypothesis, implicit paralelism, k-arm bandit

SCHEMA THEORY

Schemata

- *Individual* is a word in alphabet {0, 1}
- *Schema* is a word in alphabet {0, 1,*}
  - (* = don't care)
- Schema represents a set of individuals
- Schema with r * represents 2^r individuals
- Individual with length m is represented by 2^m schemata
- There is 3^m schemata of length m
- In population of n individuals there is between 2^m and $n.2^m$ schemata represented

Properties of schemata

- *Order* of schema S: o(S)
  - Number of 0 and 1 (*fixed* positions)
- *Defininig length of* schema S: d(S)
  - Distance between the first and the last fixed position
- *Fitness* of the schema S: F(S)
  - Average fitness of the individuals in a population that correspond to the schema S
  - Note that fitness of S depends on the context of a population.

The schema theorem

- *Short (w.r.t. defining length), above-average (w.r.t. fitness), low-order schemata increase exponentially in successive generations of GA. (Holland)*
- Building blocs hypothesis:
  - GA seeks suboptimal solution of the given problem by recombination of short, low-order above-average schemata (called building blocks).
  - "just as a child creates magnificent fortress through arrangement of simple blocks of wood, so does a GA seek near optimal performance ..."

Proof of TST

- Population P(t), P(t+1), ... n individuals of length m
- What happens to a particular schema S during:
  - Selection
  - Crossover
  - Mutation
- C(S,t) ... Number of individuals representing schema S in population P(t)
- We will estimate C(S,t+1) in three steps

Proof of TST

- Selection:
  - An individual probability of selection is:

  $p_s(v) = F(v) / F(t)$, where $F(t) = \Sigma F(u)$, {u in P(t)}
  - Probability of selection od schema S:

  $p_s(S) = F(S) / F(t)$
  - Thus: $C(S,t+1) = C(S,t)\ n\ p_s(S)$
  - Or equivalently: $C(S,t+1)=C(S,t)\ F(S)/F_{prum}(t)$

  Where $F_{prum}(t)=F(t)/n$ … is average fitness in P(t)

Proof of TST

- ... Still selection:
  - So, we have: $C(S,t+1)=C(S,t)\ F(S)/F_{prum}(t)$
  - If the schema were "above-average" of e%:
  - $F(S,t)=F_{prum}(t) + e\ F_{prum}(t)$, for t=0, ...
  - $C(S,t+1)=C(S,t)\ (1+e)$
  - $C(S,t+1)=C(S,0)\ (1+e)^t$
  - I.e. the number of above-average schemata grows exponentially (in consecutive populations (and with selection only)).

Proof of TST

- Crossover:
  - Probability that a schema will be destroyed / survive a crossover:
  - $p_d(S) = d(S)/(m-1)$
  - $p_s(S) = 1 – d(S)/(m-1)$
  - Crossing over with probability p_c:
  - $p_s(S) >= 1 – p_c \cdot d(S) / (m-1)$
- Selection and crossover together:
  - $C(S,t+1) >= C(S,t) \cdot F(S)/F_{prum}(t) [1- p_c \cdot d(S) / (m-1)]$

Proof of TST

- Mutation:
  - 1 bit will not survive: p_m
  - 1 bit will survive: $1 – p_m$
  - A Schema will survive ($p_m << 1$):
  - $p_s(S) = (1 – p_m)^{o(S)}$
  - $p_s(S) = \ldots$ roughly $\ldots = 1 – p_m.o(S)$, for small p_m
- Selection, crossover and mutation together:
- $C(S,t+1) >= C(S,t).F(S)/F_{prum}(t)\ [1-p_c.d(S)/(m-1)-p_m.o(S)]$
- QED.

Consequences of TST and BBH

- Encoding matters
- Size matters
- Premature convergence harms
- When GA sucks:
  - (111*******), (********11) are above-average
  - But F(111*****11) << F(000*****00)
  - Ideal is (1111111111); GA has hard times finding it
  - The selection condition might be improved

Implicit paralelism

- GA works with individuals, but implicitly it evolves much more schemata: 2^m to $n.2^m$.
- But how many schemata is processed efficiently:
  - Holland (and others): (Under certain circumstances, such as $n = 2^m$, schemata stay above-average, ...) Number of schemata that really grow exponentially is in the order of n^3.
- It was jokingly commented as the only case where combinatorial explosion is on our side.

Exploration vs. Exploitation

- Original Holland motivace: GA is "adaptive plan" looking for equilibrium between:
  - *exploration* (finding new areas for search)
  - *exploitation* (utilizing current knowledge)
- Just exploration: random walks, not utilizing previous knowledge
- Just exploitation: stucking in local optima, rigidity

1-armed bandit

2-armed bandit

- N coins, 2-armed bandit (arms payoffs have expected values m_1, m_2 and variances s_1, s_2). N-n coins is allocated to the better arm, n coins to the worse one.
- Goal: to maximize outcome / to minimalize loss.
- Analytical solution: to allocate exponentially more trials to the currently winning arm
- $N-n^* = O(\exp(c\, n^*))$;
  - c depends on m_1, m_2, s_1, $s_{2;}$ and n* is the optimal value

Bandit and SGA

- GA also allocates exponentially more trials (slots in population) to the more successful schemata
- It thus solves the exploration vs. exploitation problem in the optimal way
- Schemata plays many multi-armed bandit games
  - The winning prize is number of slots in popuplation
  - It is hard to estimate the fitness of a scheme
  - First people thought that SGA plays 3^m –armed bandit,
  - Where all schemata are competing arms …

… but it's complicated

- Actually, much more games is played in parallel
- Schemata "compete" for "conflicting" fixed positions in a gene
- Schemata of order k always compete for those k fixed positions – they play 2^k –armed bandit
- So, the best of those games get the exponential slots in population
- But, it depends if we can estimate the fitness of a scheme in a particular population well (which can be a problem)

Thus, a bad task for SGA is ...

- = 2; for x ~ 111*... *
- f(x) = 1; for x ~ 0*...*
- = 0; otherwise.
- For schemata we now have:
  - F(1*...*) = 1/2 ;
  - F(0*...*) = 1
- But, the SGA estimates F(1*...*) ~ 2,
- Because schemata 111*...* will be much more common in a population
- SGA here does not sample schemata independently, so it does not estimate their real fitness.

Problems

- The arms in bandit are independent, but the SGA does not sample schemata independently
- Selection does not work ideally, as in the TST, it is dynamic and it has statistical errors.
- SGA maximizes its on-line performance, they should be suitable for adaptive tasks (It is a pitty to stop a running SGA ;-)
- (Paradoxically, maybe) the most common application of GA is to let them “only” find the one best solution.

Static BBH

- *Grafenstette, 91: People consider that GA converges to solutions with actual statistic average fitness; and not (as it really happens) to those that exist in populations, i.e. with the best observed fitness*
- Then, people can be disappointed:
  - Collateral convergence
  - Large fitness variance

Collateral convergence

- When GA converges somewhere, the schemata are no longer sampled uniformly, but with a bias
- If, e.g. a sheme 111***...* is good, it will spread in a population after few generations, i.e. almost all individuals will have this prefix.
- But then, almost every sample of a scheme ***000...* are also samples of a scheme 111000*...*.
- Thus, the GA will not estimate F(***000*...*) correctly.

Large fitness variance

- GA will not estimate fitness of a scheme well in the case if the static average fitness has a large variance.
- Such as the scheme 1*...* from our evil example.
- The variance of its fitness is large, so the GA will probably converge to those parts of a search space where the fitness is big.
- Which in turn will bias further sampling of the scheme. So, the static fitness is not estimated well, again.

Integer and floating point representations operators, selection

REPRESENTATION AND OPERATORS

Encoding

- **Binary**
  - Classic (Holland)
  - There are nice theoretical results (better than schemata theory, we will see next semestr)
  - *Holland argumen: binary strings of length 100 are better than decimal of leghth 30 because they encode roughly the same information but have more schemta (2100>230).*
  - But we know schemata are not that important as Holland thought
  - The important factor is that binarz encoding is sometimes unnatural for a given problem.

Other encodings

- Alphabets with more symbols
- Integers
- Floating point

- Yet another examples:
  - Permutations,
  - Trees (programs),
  - Matrices,
  - Neural networks (different ways),
  - Finite automata
  - Graphs,
  - A-life agents …

Selection - overview

- **Roulette-wheel selection**
  - traditional, fitness-proportional
- **SUS (stochastic universal sampling)**
  - Just one random position in a roulette wheel, other positions are shifts over angle 1/n
  - „more fair roulette“ – why?
- **Turnament**
  - k-tournament − comparing k randomly selected individuals, the winner is chosen by selection
  - Typically, k is a small number, like 2, 3, 5
  - Can be used in cases where fitness is not explicitly given (a game is played, or a simulation is involved)

Integer encoding

- Mutation:
  - „unbiased“ – new random value from the whole domain
  - „biased“ – new value represents a random shift (normal distribution) from the original value
- Crossover:
  - One-point, multipe-point, …
  - Uniform – in every gene we throw a coin from which parent the value is chosen
  - Beware of ordinal representations in cases where the order does not make sense (then, probably, the biased mutation does not make sense)

Floating point encoding

- Historically, the first attempts were encoding real numbers into bit-string representations
- Not used often today, except for the cases when a limited precision makes good sense (compression of a search space, explicit control over the accuracy of the representation)
- Common practice today is to encode real values as floating point representation, and the operators take this into account

Floating point operators

- Mutation
  - biased
  - Unbiased
- Crossover
  - Structural
    - One-point, uniform, ...
  - Arithmetic
    - Combination of values

Arithmetic crossover

- Simple average of parents‘ values
- Variants:
  - Some other convex combination:
    - $z = a*x + (1-a)*y$, where $0<a<1$
  - How many values from an individual to cross:
    - Typically all of them
    - Sometimes just one chosen at random
    - Sometimes a combination with 1-point crossover

Prisoners and their dilemma, Nash, von Neumann, Axelrod, Dawkins

EVOLUTION OF COOPERATION

ROMAN NERUDA: EVA1 - 2014/15

Altruism vs. darwinism ?

- Darwinism is inherently competitiv – survival of the fittest
  - social darwinism − backing the *laissez-faire („let it be“)* capitalism
  - Andrew Carnegie, The Gospel of Wealth, 1900 *While the law of competition may be sometimes hard for the individual, it is best for the race, because it ensures the survival of the fittest in every department. We accept and welcome, therefore, as conditions to which we must accommodate ourselves, great inequality of environment; the concentration of business, industrial and commercial, in the hands of the few; and the law of competition between these, as being not only beneficial, but essential to the future progress of the race.*
- But there is a lot of cooperation both in nature and society
- The main problem of evolutionary (social) biology:
- **How can altruistic behavior be evolved, when it (by definition) decreases a fitness of a n individual?**

Theories of evolution of altruism

- Group selection
  - Evolution can work on groups of individuals (Darwin)
  - How to explain individuals who cheat and do not help
- Kin selection
  - Preservation of almost identical genes in close relatives
  - How to explain altruism of strangers, even other species
- Dawkins, selfish gene
  - The unit of evolution is a gene, not an individual
  - Wilson: „the organism is only DNA's way of making more DNA.“
- Trivers, 1971: reciprocal altruism
  - Mutual benefits for both organisms (even different species)
  - Shadow of the future, paralell with iterated prisoners dilemma

Prisoner's dilema

i/j	D	C
D	2 / 2	0 / 5
C	5 / 0	3 / 3

i/j	D	C
D	P / P	S / T
C	T / S	R / R

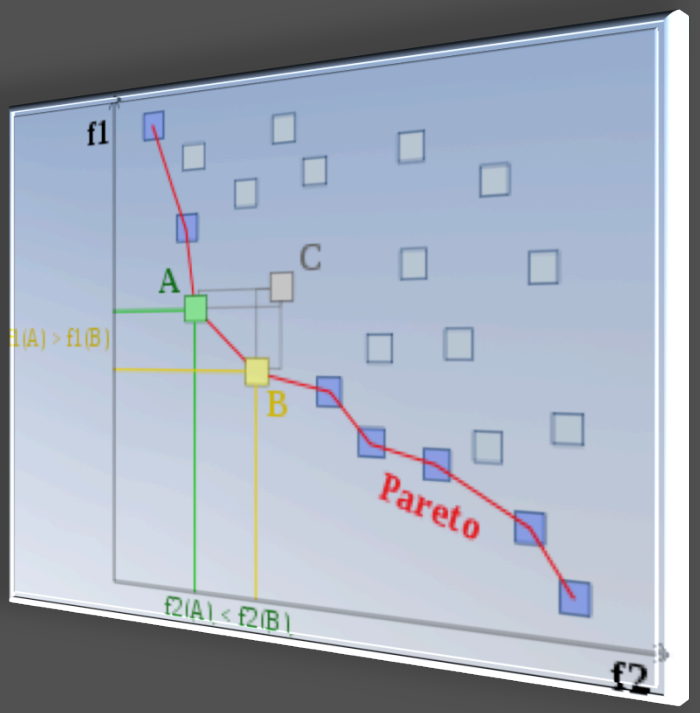
- *Temptation > Reward > Penalty > Suckers payoff*
- *R>P: mutual cooperation is better than mutual deception*
- *T>R a P>S: deception is a dominant strategy for both players*
- (50s- RAND corp.)

Nash

- A strategy s is *dominant* for agent *i*, if it gives better or the same result than any other strategy of an agent i against all strategies of agent j
- Strategies s_i and s_j are in *Nash equilibrium*, if:
  - If agent *i* plays strategy s_i, agent *j* does best with strategy s_j
  - If *j* plays s_j, i does best with s_i
- Or, s_i and s_j are the best mutual answers to each other
- This is Nash equilibrium of pure strategies
    - But not every game has a Nash equilibrium in pure strategies
    - And some games have more Nash equilibria

Nash and Pareto

- *Mixed strategies* – random selection among pure strategies
  - *Nash theorem*: Every game with finite number of strategies have Nash equilibrium in mixed strategies.
- The solution is *Pareto-optimal/ efficient*
  - If there is no other strategy which would improve agent outcome without worsening some other agent outcome
  - The solution is not Pareto-efficinet: if an outcome of one agent cen be improved without decreasing other agent‘s outcome



Thus …

- For rational agents there is no dilemma/or is there?
  - DD is Nash equiilibrium
  - DD is the only solution that is not Pareto-optimal
  - CC is a solution maximizing common outcome
- Tragedy of the commons
- What is rational, and are people rational?
- Shadow of future – iterated version – Axelrod

Iterated prisoner‘s dilemma

- Players play more games, they remember the results/ acitons of the oponent, and can modify their strategies according to the history
- *T>R>P>S,*
- ***2R>T+S –** it does not pay off to alternate C and D*
- If the game is played N-times (and the players know the N) it can be proved by induction, the best strategy is „deceive all the time“.

Axelrod tournaments

- **The first tournament:**
  - 14 strategies plus RANDOM, 200 games, everybody played with everybody (ncludit itself), 5x repeat
- TFT = Tit For Tat strategy
  - Start cooperate, then copy oponent‘s moves
- **The second tournament:**
  - 62 strategiíes– everybody knew the results of previous tournament –TFT wins again
- The third „ecological“ tournament
  - Resembling the generations of GA, initial population was the second tournament strategies, there were 1000 generations
  - The number of individuals in the next generation was proportional to number of victories in the previous generation
  - Aaaaand, the TFT wins again!

What does it mean for strategies?

- 4 important properties of successful strategies:
  - Niceness – do not deceive first
  - Provocability – quickly punish deception
  - Forgiveness – but quickly calm down
  - Clarity – be simple, so others understand you
- There is not a single strategy that would win against all strategies
- It is necessary to be successful against very diverse strategies (ALL-D, TFTT, RANDOM, TRIGGER)
- It is also good to learn play well against itself
- Attempts to beat TFT by more deception did not help

What does it mean for cooperation?

- In environments that support cooperation …
  - Payoffs favor cooperation,
  - There is a big probability of iterated PD (shadow of the future)
- … the cooperation is usually evolved
  - But not always, such as in the ALL-D world
- Rationality, intelligence, consciousnes, … is not necessary for cooperation, just bigger fitness values
- Initial cooperation can emerge at random, and then it can survive

Twenty years after

- In environments with noise, the Pavlov strategy (win-stay, lose-shift) is successfu
  - If the payoff R or P => C,
  - if T or S => D
- After 20 years the tournament was repeated with more strategies from each team
  - The winning strategies were cooperating as a team
  - Few moves (10) tahů to recognize the oponent, then all strategies helped one father strategy from the team to get better score
  - The teams were even fighting the organizers (false teams to get more slots in the tournament …)

Motivation, population cycle, floating point mutations, meta-evolution

EVOLUTIONARY STRATEGIES

Evolutionaty strategies

- Rechenberg, Schwefel, 60s
- Optimization of real function of many parameters
- 'evolution of evolution'
- Evolved individual:
  - *Genetic parameters* - affecting the behavior
  - *Strategic parameters* - affecting evolution
- New individual is accepted only if it is better
- More individuals as parents
- Todays most successful (and complex) is CMA-ES (correlation matrix adaptation-ES)

ES notation

- Important parameters:
  - *M* number individuals in population
  - *L* number of new individuals
  - *R* počet 'rodičů'
- Special selection related notation:
  - *(M+L)* ES – *M* individuals to a new generation is selected from *M+L* old and new individuals
  - *(M,L)* ES – *M* individuals to a new generation is selected only from *L* new individuals
    - Usually, the (M,L) strategies are more robust − less prone to stuck in local optima
- The individual: $C(i)=[G_n(i),S_k(i)]$, *k=1, or n,* or *2n*

ES population cycle

- *n=0*; Initialize at random a population P_n of *M* individuals
- Evaluate the fitness values of individuals in P_n
- Until the solution is not good enough:
  - Repeat *L* times:
    - choose *R* parents,
    - Cross them over, mutate, evaluate the new individual
  - Choose *M* new individuals (depending on the ES type)
  - *++n*

ES individual and mutation

- $C(i)=[G_n(i),S_k(i)]$
- S_k are standard deviations of biased floating point mutations
- k=1:
  - One common std dev for all evolved parameters G's
- k=n:
  - Non-correlated mutations, n individual normal distributions
  - Each parameter has its own std dev
  - Geometricly, the mutations are within an ellipse parallel to axes
- k=2n:
  - Rotations are also included, the ellipse is not parallel to axes
  - correlated mutations, they correspond to mutations from n-dimensional normal distribution
  - n parameters for rotations, n for std devs 2n

ES mutations

- Genetic parameters:
  - Adding random number from normal distrbution with corresponding deviation, and rotation, respectivelly
- Standard deviations:
  - Increase or decrease according to the success of the mutation
  - Originally, the so-called 1/5 rule (heuristic, „the best case is when the mutation has 20% success rate“, thus, the std dev is increased for lower success rates, and decreased when the success rate is higher
  - More common now is to add a random number drawn from N(0,1)
- Rotation:
  - Add a random number drawn from N(0,1)

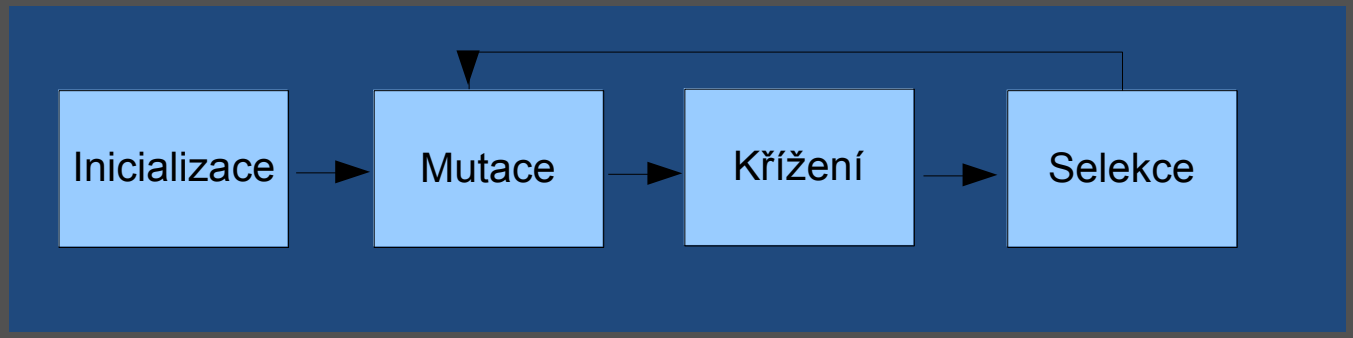
ES crossover

- Uniform
- „Gang bang“ of more parents
  - Local (R=2)
  - Global (R=M)
- Two versions:
  - Discrete
  - Arithmetic (average)

Alternative, geometrically motivated EVA

DIFFERENTIAL EVOLUTION

DE – scheme and initialization



- **Inicialization**: random parameter values
- **Mutation**: „shift“ according to the others
- **Crossover**: uniform „with a safeguard“
- **Selection**: comparison and possible replacement by a better offspring

Mutation

- Every individual in a population undergoes mutation, crossover, and selection
- For an individual $\boldsymbol{x}_{i,p}$ we choose three different individuals $\boldsymbol{x}_{a,p}$, $\boldsymbol{x}_{b,p}$, $\boldsymbol{x}_{c,p}$ at random
- Define a donor $\boldsymbol{v}$: $\boldsymbol{v}_{i,p+1} = \boldsymbol{x}_{a,p} + F.(\boldsymbol{x}_{b,p} - \boldsymbol{x}_{c,p})$
- F is a mutation parameter, a value from interval <0;2>

Crossover

- Uniform crowwover of original individual with a donor
- Parameter *C* controls the probability of a change
- At least one element must come from a donor
- Probe vector $u_{i,p+1}$:
- $\boldsymbol{u}_{j,i,p+1} = \boldsymbol{v}_{j,i,p+1}$; iff $rand_{ji} <= C$ or $j=I_{rand}$
- $\boldsymbol{u}_{j,i,p+1} = \boldsymbol{x}_{j,i,p+1}$; iff $rand_{ji} > C$ and $j \neq I_{rand}$
- $rand_{ji}$ is pseudorandom number from <0;1>
- I_{rand} pseudorandom integer from <1;2; ... ; D>

Selection

- Compare fitness of $\boldsymbol{x}$ and $\boldsymbol{v}$, select the better:
  - $\boldsymbol{x}_{i,p+1} = \boldsymbol{u}_{i,p+1}$; iff $f(\boldsymbol{u}_{i,p+1}) <= f(\boldsymbol{x}_{i,p})$
  - $\boldsymbol{x}_{i,p+1} = \boldsymbol{x}_{i,p}$; otherwise
  - for $i=1,2, \ldots , N$
- Mutation, crossover, and selection is repeated until some termination criterion is satisfied (typically, the fitness of the best individual is good enough)

Individual is a particle floating in a swarm in the fitness landscape

PARTICLE SWARM OPTIMIZATION

PSO

- Population-based search heuristic
- Eberhart, Kennedy, 1995
- Inspiration of swarms of insect/fish
- Individual is typically a floating point vector
- It is called a ***particle***
- No crossover
- No mutation as we know it
- Individuals are moving in a swarm through their parameter space
- The algorithm is using local and global memory:
  - pBest − each particle remembers a position with the best fitness
  - gBest − best pBest among all particles

PSO algorithm

- Initialize each particle
- Do
- Foreach particle
- Compute fitness of particle
- If the fitness is better than the best fitness seen so far (pBest)
- pBest := fitness;
- End

- Set gBest to the best pBest
- Foreach particle
- compute the speed of particle by equation (a)
- update position of particle by equation (b)
- End
- While maximum iterations or minimum error not satisfied

PSO movement equations

- ***v*** *: =* ***v*** *+*

  *+ c1 * rand() * (**pbest** - **present**) +*

  *+ c2 * rand() * (**gbest** - **present**)* *(a)*
- ***present*** *=* ***persent*** *+* ***v*** *(b)*
- ***v*** is particle speed, ***present*** is particle position
- ***pbest*** best position of a particle in history
- ***gbest*** best global position in history
- *rand()* random number from (0,1).
- *c1, c2* constants (learning rates) often *c1 = c2 = 2*.

PSO discussion

- Common with GA:
  - Start with random configuration, have a fitness, use stochastic update methods
- Different from GA:
  - No genetic operators
  - Particles have memories
  - The exchange of information goes only from the better particles to the rest

Michigan vs. Pittsburg, machine learning, reinforcment learning

EVOLUTIONARY MACHINE LEARNING

Machine learning – a subset

- Learn rules based on the training examples
  - Data mining
  - Expert systems
  - Agent, robots learning (reinforcement learning)
- Basic evolutionary approaches:
  - **Michigan** (Holland): individual is one rule
    - Holland LCS: learning classifier systems
  - **Pittsburgh**: individual is a set of rules

Michigan

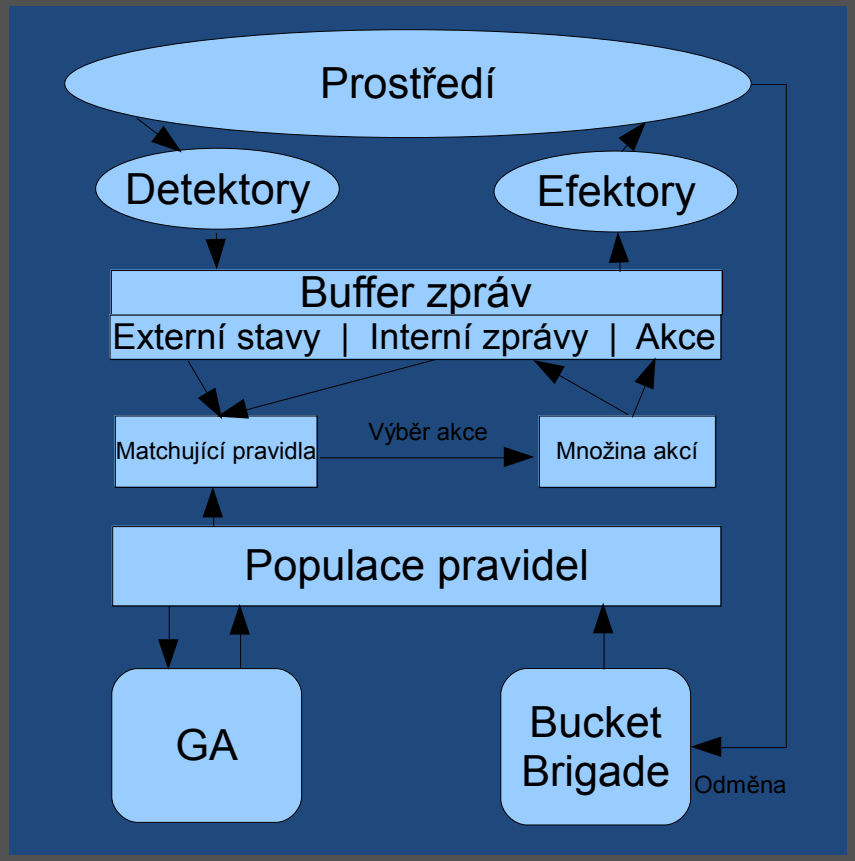
- Holland in 80s: learning classifier systems
- The individual is a rule
- The whole population works as an expert or control system
- The rules are simple:
  - Left-hand side: feature is true/not/don‘t care (0/1/*)
  - Right-hand side: action code or classification category
- Rules have weights (reflecting their success)
- The weight makes their fitness
- The evolution does not have to be generational

Michigan - LCS

- Evolution happens only from time to time and/or on part of population
- The problem of reactivness (lack of inner memory)
  - The right-hand side of the rule contains – besides the action/classification code – other inner features, called „messages“
  - The left-hand side of the rule has special features to intercept the messages, called „receptors“
  - The system has a buffer of messages and it has to realize an algorithm to distribute a reward among chains of rules

LCS – bucket brigade

- Only some rules lead to actions that trigger reward from the environment,
- The reward should be distributed to the chain of successful rules leading to the reward
- Rules have to give up part of their strenght (like paying money to take part in the action) if they compete for a chance to be applied
- The technical way it is done is called **Bucket brigade algoritm**
- In practice it is difficult to ballance the economy of rules, hardly used today



Z(ero)CS

- (Wilson, 1994) simplify LCS
  - No internal messages
  - No complicated mechanism of reward redistribution
- Rules are just bitmap (and *) representations:
  - IF(inputs) THEN (outputs)
- Cover operator:
  - If there is no rule for current situation/example, it is generated ad hoc
  - Randomly some * are added and a random output is selected

ZCS contd.

- How the reward is disrtibuted / the strength of rules is modified:
  - Rules not applicable to given situation: nothing
  - Rules applicable to input but with different output: decrese the strenght by multiplying by constant $0<T<1$
  - All rules' strenghts are decreased by a small constant B
  - This amount id distributed uniformly among the rules that answered correctly in the *previous* step (decreased by a factor $0<G<1$)
  - Finally, the answer of the system is decreased by B and uniformely distributed among rules that answerd correctly in *this* step

XCS – improved ZCS

- Cons of ZCS:
  - ZCS does not tend to evolve a complete rule system covering all cases
  - Rules at the beginning of the chains are seldom rewarded and they are not surviving
  - Rules leading to actions with small rewards can die off too, although they are important
- XCS:
  - Separate fitness from expected outcome/reward of the rule
  - Base fitness on the specificity of the rule

Pitt

- Individuals are sets of rules, complete systems
- The evaluation is more compicated
  - Rule priorities, conflicts
  - False positives, false negatives
- Genetic operators are more complicated
  - Typically, dozen or more operators working os sets of rules, individual rules, terms in the rules, ...
- Emphasis on rich domain representation (sets, enumerations, intervals, ...)

GIL, example of Pitt aproach

- Binary classification tasks
- The individual classifies implicitly to one class (no right-hand side of the rules)
- Each individual is a disjunction of *complexes*
- Complex is conjunction of *selectors (from 1 variable)*
- Selector is a disjunction of values from the variable domain
- Representation by a bitmap:
    - ((X=A1)AND(Z=C3)) OR((X=A2)AND(Y=B2))
    - [001|11|0011 OR 010|10|1111]

GIL contd.

- Operators on the individual level:
  - Swap of rules, copy of rules, generalization of rule, deletion of rule, specialization of rule, inclusion of one positive example to the rule
- Operators on the complex level:
  - Split of complex on 1 selector, generalization of selector (replacing by 11...1), specialization of generalized selector, inclusion of one negative example
- Operators on selectors:
  - Mutation 0<->1, extension 0->1, reduction 1->0,

Multi-Objective Evolutionary Algorithms (MOEA), Paretova fronta, NSGA II

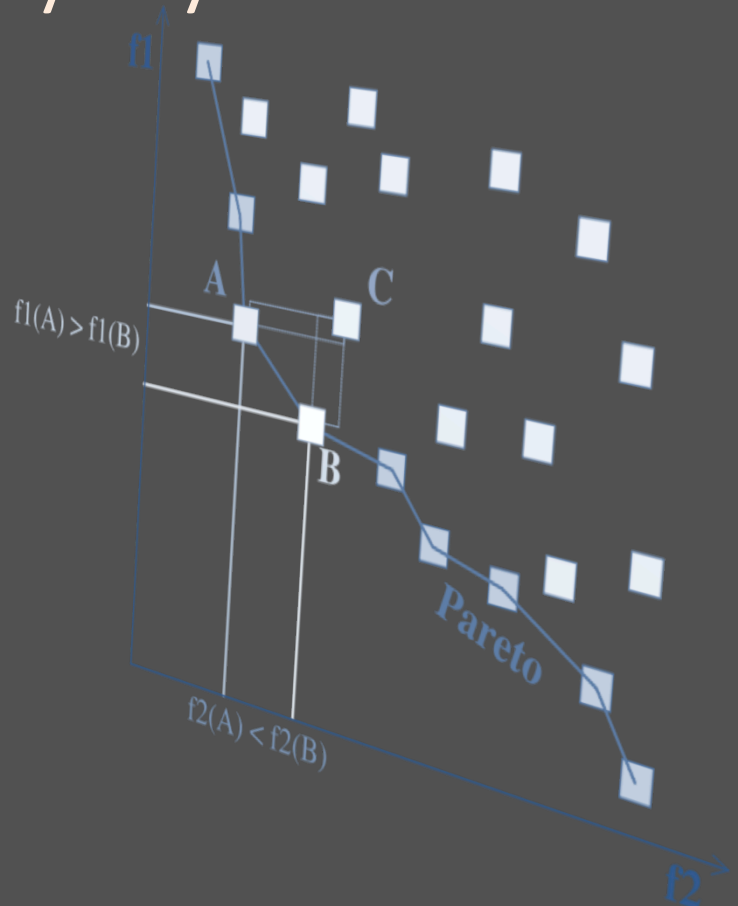
MULTI-OBJECTIVE OPTIMIZATION

Problem

- Instead of one fitness (objective function), there is a vector of them f_i, $i=1...n$
- For the sake of simplicity, we consider minimization case, so we try to achieve minimal values of all *fi*, which is difficult
- Definitions of dominance (of individual, or a solution):
  - Individual *x* **weakly dominates** individual *y*, iff *fi(x)<=fi(y),* pro *i=1..n*
  - *x* **dominates** *y*, iff it weakly dominates him, and there exists *j: fj(x) < fj(y)*
  - *x* and *y* are **uncomparable**, when neither *x* dominates *y*, nor *y* dominates *x*
  - *x* **does not dominate** *y*, if either weakly dominates *x,* or they are uncomparable

Pareto front

- **Pareto front** is a set of individuals not dominated by any other idividual



The simple way

- How to solve MOEA in a simple (simplistic?) way:
- Aggregate the fitness:
  - i.e. weighted sum of all *fi*, resulting in one value of *f*
  - And solve it as a standard one-objective optimization
  - This one is sometimes, in the context of MOEA, called SOEA (single objective EA), but is is nothing new to us, actually we were doing only SOEA so far
- Nevertheless, we do not know how to set weights for individual *fi*'s.

VEGA (Vector Evaluated GA)

- One of the first MOEAs, 1985
- Idea:
  - Population of *N* individuals is sorted according to each of the *n* objective functions
  - For each *i* we select *N/n* best individuals w.r.t. *fi*
  - These are crossed over, mutated and selected to next generation
- This approach in fact, has lots of disadvantages:
  - It is difficult to preserve a diversity of the population
  - It tends to converge to optimal solutions for individual objectives *fi*

NSGA (non-dominated sorting GA)

- 1994, an idea of dominance is used for fitness
- This still does not guarantee sufficient spred of population, it must be dealt with some other way (niching)
- Algorhitm:
  - Population P is divided into consequently contructed fronts F1, F2, ...
    - F1 is a set of all non-dominated individuals from P
    - F2 is a set of all non-dominated individuals from P-F1
    - F3 … from P-(F1 disjuncted with F2)
    - …

NSGA contd.

- For each individual we compute a **niching faktor**, as a sum of *sh(i,j)* over all individuals j from the same front, where:
  - *sh(i,j) = 1-[d(i,j)/dshare]^2*, for *d(i,j)<dshare*
  - *sh(i,j) = 0* otherwise
    - *d(i,j)* is distance *i* from *j*
    - *dshare* is a parameter of the algorithm
- Individuals from the first front receive some„dummy“ fitness, that is divided by a niching factor
- Individuals from the second front recieve a dummy fitness smaller that the fitness of the worst individual from the first front, and it is again divided by their niching factor
- ... For all fronts

NSGA II

- 2000, repairing some drawbacks of NSGA:
  - Necessity to set the right dshare value
  - Non-existence of elitism
- **Niching**
  - *Dshare* a *niche count* is replaced by a **crowding distance:**
  - This is a sum of distances to the nearest neighbours
  - The best individuals w.r.t. each *fi's* have crowding distance set to infinity
- **Elitismus**
  - Old and new populations are joined, sorted, the better part goes to next generation

NSGA II contd.

- **Fitness:**
  - Each individual has a number of non-dominated front it is in, and a crowding distance
  - When comparing two individuals, first a front is considered (smaller is better), and in case of the same front, their crowding distance is considered (bigger is better)
  - And in fact, no fitness is really computed, just these two numbers are compared in a tournament selection
- And now we have an improvement – NSGA III

EVA solves NP-hard problems, TSP, permutaion representations

COMBINATORIAL OPTIMIZATION

EVA solves hard tasks

- 0-1 knapsack problem
  - Simple encoding
  - Problematic fitness
  - Standard operators
- Travelling Salesman problem (TSP)
  - Simple fitness
  - Problematic encoding and opertators (crowwover, really)
- Scheduling, planning, transportation problems ...

Knapsack

- Given:
  - A knapsack of capacity CMAX
  - N items,
  - each have a price v(i)
  - and a volume c(i)
- The task is to choose items such that:
  - Maximize a sum v(i)
  - At the same time we squeeze them into a knapsack, i.e. Sum of c(i) <= CMAX

Knapsack

- Encoding − a bitmap:
  - 0110010 – take items 2,3 and 6
  - Trivial almost
  - But the individuals might not satisfy the CMAX condition
- Operators:
  - Simple crossover, mutation, selection
- Fitness: has two parts:
  - max [sum of v(i)] vs. min [CMAX – sum of c(i)]

Knapsack

- So, we have a multi-objective optimization:
  - Either weight em and add em
  - Or use your favourite MOEA from previous chapter
  - Or, change the encoding in a clever way:
    - 1 means: PUT the item in the knapsack UNLESS the capacity is not exceeded
    - This way we achieve a nice property that with such a decoder all strings in fact represent a valid solution

Travelling salesman

- N cities, tour them with minimal cost
- Fitness is clear cost of the trip
- Reprezentations are many
  - Variants of vertex-based
  - Edge-based, ...
- Operators are heavily dependent on representation
  - Crossover allows to use heuristics we might have to solve the TSP

Adjacency representation

- Path is a list of cities, city j is at position I iff there is an edge from i to j
- Ex:
  - (248397156) corresponds to 1-2-4-3-8-5-9-6-7
- Each path has 1 representation, some lists do not generate valid paths
- Not very intuitive
- Classical crowwover does not work
- But schemata do:
  - E.g. (*3*...) means all paths with 2-3 edge
- Do not use it.

Ordinal (or buffer) representation

- Motivovation was to use the standard 1-point crossover
  - Let us have a buffer of vertices, maybe just ordered, the encoding is in fact a position of a city in this buffer
  - When a city is used, it is deleted from a buffer
- Ex:
  - Buffer (123456789), and path 1-2-4-3-8-5-9-6-7 is repesented as (112141311)
- Do not use it either.

Path (or permutation) representation

- Probably a first idea of most people
- Permutation representation is important as natural for many other tasks, as well.
  - path 5-1-7-8-9-4-6-2-3 is represented as (517894623)
- The crossover does not work
- So, the main problem with this representation is to propose a crossover operator that produces correct individuals, and represents some idea about how a good solution should look like.
  - PMX, CX, OX, ...

PMX

- Partially mapped crossover (Goldberg)
- Preserve as many cities on their positions from the individuals as you can.
- 2-point
- (123|4567|89) PMX (452|1876|93) :
  - (…|1876|..) (…|4567|..)
  - and a mapping 1-4 8-5 7-6 6-7
  - Can be addedd (.23|1876|.9) (..2|4567|93)
  - According to the mapping
    - (423|1876|59) (182|4567|93)

OX

- Order crossover (Davis)
- Preserve relative order of cities in the individuals
- (123|4567|89) OX (452|1876|93) :
  - (...|1876|..) (...|4567|..) rearrange the path from the second crossover point
  - 9-3-4-5-2-1-8-7-6
  - Delete crossed over cities from 1, remains: 9-3-2-1-8
  - Fill the first child: (218|4567|93)
  - Similarly, the second child: (345|1876|92)

CX

- Cyclic crossover (Oliver)
- Preserve the absolute position in the path
- (123456789) CX (412876935)
  - First position at random, maybe from th first parent: P1=(1.....),
  - Now we have to take 4, P1=(1..4....), then 8, 3 a 2
  - P1=(1234...8.), can't continue, we fill from the second parent
  - P1=(123476985)
  - Similarly P2=(412856739)

ER

- Edge recombination (Whitley et al)
- Observation: all previous crossovers preserve only about 60% of edges from both parents
- The ER tries to preserve as many edges as possible.
  - For each city make a list of edges
  - Start somewhere (the first city),
  - Choose cities with less edges,
  - In case of the same number of edges, choose randomly

(123456789) ER (412876935)

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

- Start in 1, successors are 9, 2, 4
- 9 looses, has 4 succ., from 2 and 4 choosing at random 4
- succ. of 4 are 3 and 5, take 5,
- Now we have (145……), and continue
- … (145678239)
- It is possible that we cannot choose an edge and the algorithm fails, but it is very rare (1-1.5% případů)

(123456789) ER2 (412876935)

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

- ER2 – improving ER
- Preserving more common edges
- Mark edges that exist twice by - #
- They are prioritized when choosing where to go.

Initialization for TSP

- Nearest neighbours:
  - Start with a ranodm city,
  - Choose next as the closest from the not chosen yet
- Edge insertion:
  - To a path T (start with an edge) choose the nearest city c not in T
  - Find an edge k-j in T so it minimizes the difference between k-c-j and k-j
  - Delete k-j, insert k-c and c-j to T

Mutation for TSP

- Inversion (!)
- Insert a city into a path
- Shift subpath
- Swap 2 cities
- Swap subpaths
- Heuristics such as 2-opt etc.
    - Take two edges, four cities, choose other two edges connecting these 4 cities

Other approaches

- (Binary) matrix representation:
  - Either 1 on position (i,j) means an edge from i to j
  - Or it means that i is before j in a path (more common)
  - Specific operators of matrix crossover:
    - Conjunction – bitwise AND and random insertion of edges
    - Disjunction – dissect into quadrants, 2 of them delete, remove contradictions, insert edges at random
- Combination with local heuristics
  - Evolutionary strategy which improves paths by “smart mutations” − heuristics like 2-opt, 3-opt

Other tasks - scheduling

- Scheduling is NP-hard:
  - Individual is a sechedule, direct matrix encoding
    - Rows are teachers, columns classes, values are codes of subjects
    - Mutation – mix the subjects
    - Crossover – swap better rows from individuals
  - Fitness
    - Fitness of a row (how a teacher is satisfied)
    - Other soft criteria and constrains about the schedule quality
  - Hard constrains
    - Must respect in operators, otherwise too many inadmissable solutions are generated
    - Teachers constrains, when, where what to teach, …

Other tasks – job shop scheduling

- Production planning
  - products o1…oN, from parts p1…pK, for each part more plans how to produce it on machines m1…mM, machines have different times for setup to a different product
  - Fitness – production time
- Encoding is critical:
  - Permutation– plan is just a permutatin of products order. Decoder must choose plans for parts. Simple representation, can use TSP-inspired crossovers. But shows not very efficient, decoder solves the complicated part, TSP operators not suitable.
  - Direct representation of individual as the complete plan – specialized and complex evolutionary operators.

Just an introduction, the cool approaches are in the EVA II

NEUROEVOLUTION

Learn neural networks by EVA

- First experiments in 80s
- Learn the parameters (weights)
- Learn the structure (architecture, connections)
- Learn weights and structure together
- Reinforcement learning tasks – when there is no supervised algorithm(robotics)
- Hybrid methods – combination of EVA with local search etc

Learn the weights

- Direct:
  - Encode the weights to a (floating point) vector,
  - floating point GA, standard operators
  - Evolutionary strategies, …
- Usually slower than specialized gradient based local algorithms, but can be robust
- Use mini batches
- Can be parallelized easily
- Can be used for reinforcement learning where there s no gradient (robotics)

Learn the structure

- Fitness = build the network, initialize at random, train, several times
- Direct encoding
  - Represent the structure as binary matrix
  - Linearize the relevant part of the matrix into a binary vector
- Gramatical encoding, Kitano
  - Individual is a representation of 2D formal grammar that are a program to create the binary matrix representing the strucrure of the neural net
  - Bold but to heavy-weight solution, not used in practice