

# EVA I

NAIL 025 – 2024

Roman Neruda

ENGLISH VERSION – -10-2024



# INTRODUCTION

Topics, sources, outlines.

# LITERATURE

- Eiben, A.E and Smith, J.E.: *Introduction to Evolutionary Computing*, Springer, 2007.
- Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs* (3ed), Springer, 1996
- Mitchell, M.: *Introduction to Genetic Algorithms*. MIT Press, 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, natural selection, simulation, objective function, roulette wheel, tournament, elitism.
- Genetic algorithms. encoding, operators, selection, crossover, mutation.
- Representational schemata, schemata theorem, building blocks hypothesis.
- Prisoner'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, learning classifier systems, bucket brigade algorithm, Q-learning.

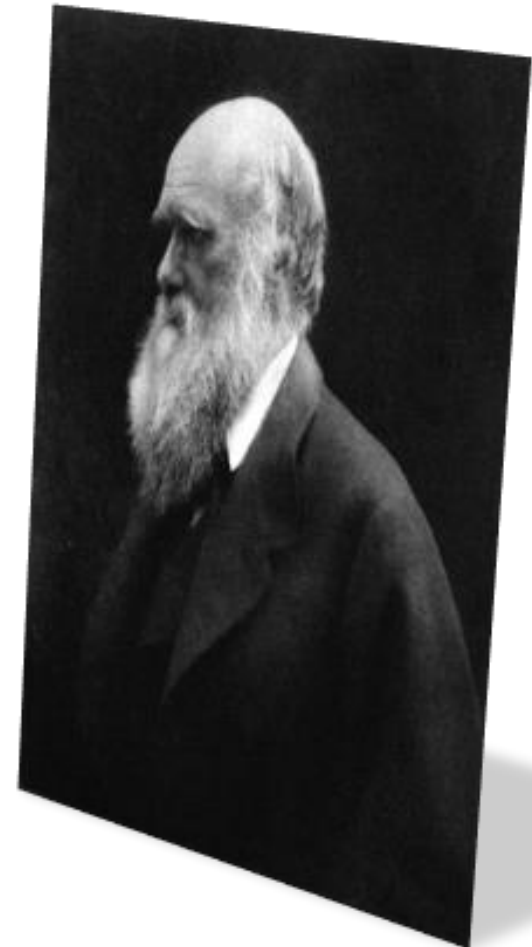


# EVOLUTIONARY ALGORITHMS

Biological motivation, basic parts

# 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 alleles, 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 and Crick – double helix structure of DNA
- Molecular-biological view:
  - How is the genetic information stored in a living organism
  - How is it inherited
- DNA consists of 4 nucleotides/bases – adenine, guanine, cytosine, thymine
- Codon – a triplet of nucleotides encoding 1 out of 23 amino acids (redundancy)
- These 23 amino acids are the basic building structure of carbohydrates in all living organisms



# MOLECULAR GENETICS

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

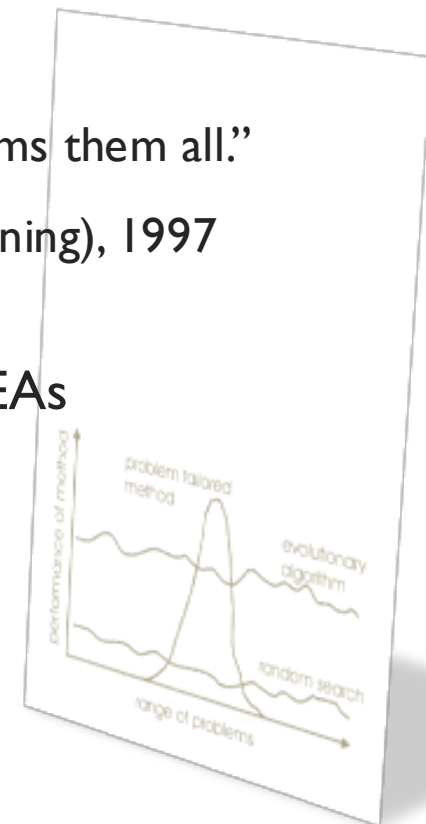


# EA - SUMMARY

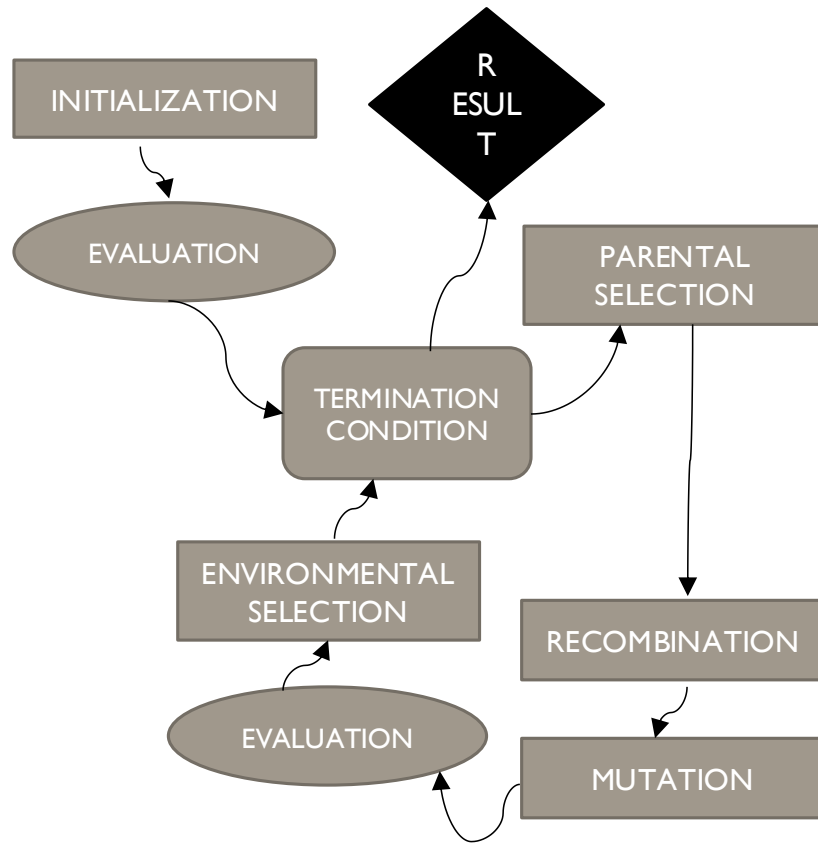
- Natural evolution: environment, individuals, fitness
- Artificial evolution: problem, candidate solutions, quality of a solution measure
- EAs are population-based stochastic search algorithms
- Recombination and mutation create variability
- Selection leads the search in the right direction

# GENERAL EA

- EAs are robust meta-algorithms
- No free lunch theorem
  - “There is no one best algorithm that outperforms them all.”
    - Wolpert, Macready, 1995 (search), 1996 (learning), 1997 (optimization)
- 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  $P(t)$  and  $P'(t+1)$



# GENETIC ALGORITHMS

- 1975 - Holland
- Binary encoded individuals
- Roulette-wheel selection
- 1-point crossover
- Bitwise mutations
- Inversion
- Schemata 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

# EVOLUTIONARY 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 undergoes 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 hardware, evolution of graph structures, ...)

# SIMPLE GENETIC ALGORITHM

Holland SGA, binary representation, operators  
and their variants

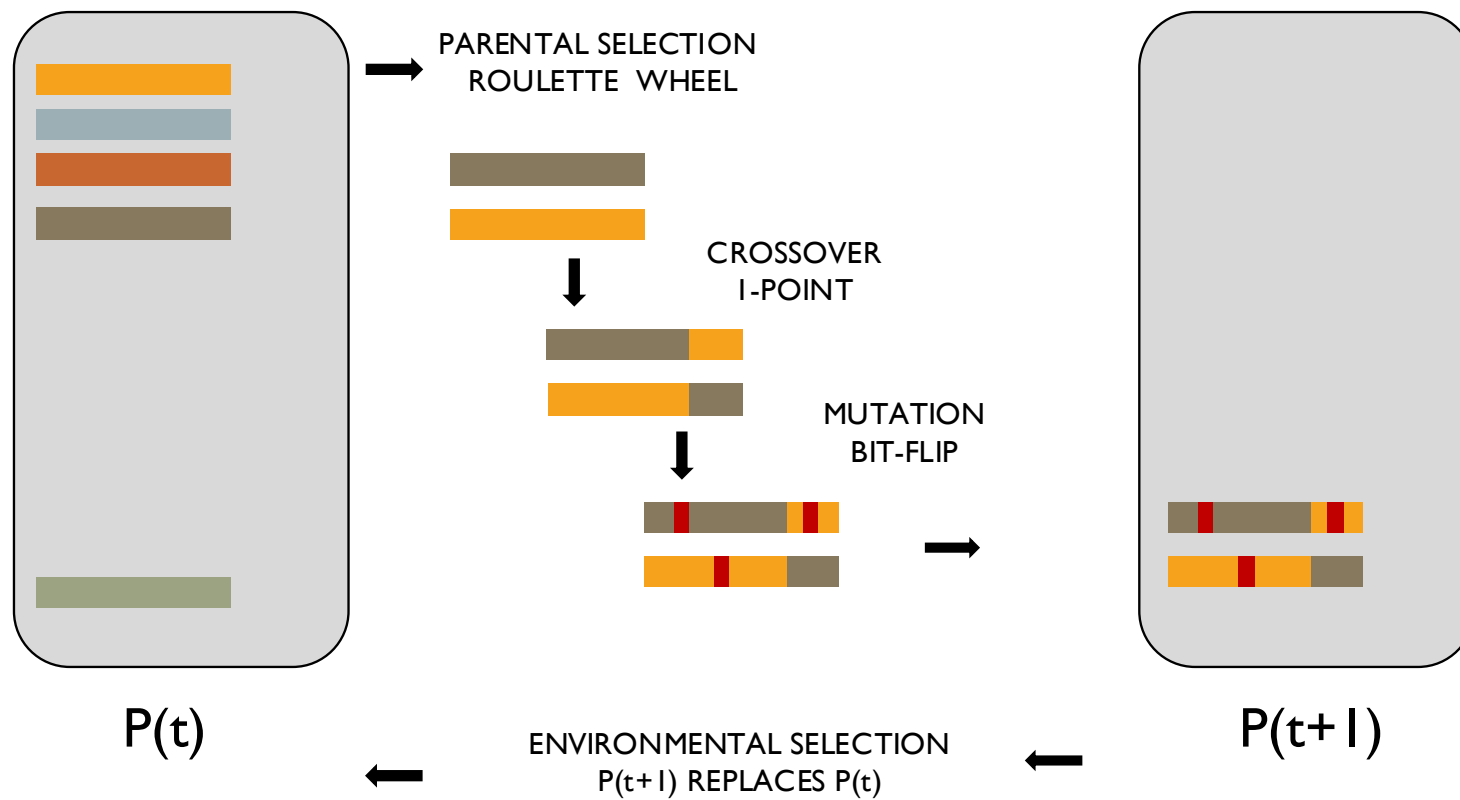
# GA

- 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  $n$  1-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  $P(t+1)$

# SGA – BASICS





# SELECTION

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

# CROSSOVER

- Crossover is the main operator in GA
- It recombines (good?) properties of parents
- It expresses our hope that recombination can lead to better fitness
- *One-point crossover*:
  - Choose crossover point at random
  - Exchange corresponding parts of individuals
  - Probability of crossover  $p_C$  typically in the range of tenths

# MUTATION

- In simple GA, mutation operator is less important,
- It acts as a mechanism against stuck in local extrema
- (On the contrary, in EP or 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 (e.g. to change 1 bit in individual on average)

# INVERSION AND OTHERS


- 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 (but wait for permutations)

# SCHEMA THEORY

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

- actually does not very well describe genetic algorithms
  - schema theory is very limited

# SCHEMATA

- *Individual* is a word in alphabet  $\{0, 1\}$
- *Schema* is a word in alphabet  $\{0, 1, *\}$   $\hookrightarrow$  set of individuals
  - (\* = don't care) One schema example:  $\boxed{110|x|*}$ 

- Schema represents a set of individuals
- Schema with  $r$  \* represents  $2^r$  individuals
- Individual with length  $m$  is represented by  $2^m$  schemata *vždy můžeme vybrat mezi fix hodnotami a \* wildcard*
- There is  $3^m$  schemata of length  $m$  *vždy lze vybrat 3 znaky na každou pozici*
- In population of  $n$  individuals there is between  $2^m$  and  $n \cdot 2^m$  schemata represented

We can compare schemata between populations  $P_t$  and  $P_{t+1}$  (where  $P_{t+1}$  comes from  $P_t$ )  $\hookrightarrow$  and then be interested what are good properties (of schema) to be selected in the next population?

Fitness of a scheme:  $\text{avg}(f(x))$  where  $x \sim S$  scheme and  $x \in P_t$

It is also expensive to evaluate fitness of an individual  $\Leftarrow$

this might be the problem.  
Some schemas might be very underrepresented

## PROPERTIES OF SCHEMATA

- Order of schema  $S$ :  $o(S)$  – number of fixed positions
  - Number of 0 and 1 (fixed positions)
- Defining length of schema  $S$ :  $d(S)$  – how much fixed positions span
  - 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)   
 *→ drawn from selection*  
*→ not getting destroyed by crossover*
- 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
  - ↳ this estimation decides what schemata are good

# PROOF OF TST

- Selection:

- An individual probability of selection is:

$$p_s(v) = F(v) / F(t), \text{ where } F(t) = \sum F(u), \{u \text{ in } P(t)\}$$

- Probability of selection of schema S:

$$p_s(S) = F(S) / F(t)$$

*previous schema's repr. prob. of selecting a scheme*

- Thus:  $C(S, t+1) = C(S, t) \cdot n \cdot p_s(S)$

- Or equivalently:  $C(S, t+1) = C(S, t) \cdot \frac{F(S)}{F_{avg}(t)}$

Where  $F_{avg}(t) = F(t)/n$  ... is the average fitness in  $P(t)$

*the better than avg., the more will be chosen*

*> this gives "expected number" of individuals in population*

# PROOF OF TST

- ... Still selection:
  - So, we have:  $C(S, t+1) = C(S, t) F(S) / F_{\text{avg}}(t)$
  - If the schema were “above-average” of  $e\%$ :
  - $F(S, t) = F_{\text{avg}}(t) + e F_{\text{avg}}(t)$ , for  $t=0, \dots$
  - $C(S, t+1) = C(S, t) (1+e)$
  - $C(S, t+1) = C(S, 0) (1+e)^t$  *— this holds only when situation is very static*
  - I.e., the number of above-average schemata grows exponentially (in consecutive populations (and with selection only)).

# PROOF OF TST

- Crossover: *→ any "\*" are not damaged by crossover*
  - Probability that a schema will be destroyed / survive a crossover: *m := size of an individual (having only m-1 "borders" to hit)*
  - $p_d(S) = d(S)/(m-1)$
  - $p_s(S) = 1 - d(S)/(m-1)$
  - Crossing over with probability  $p_c$ :
  - $p_s(S) \geq 1 - p_c \cdot d(S) / (m-1)$  *→ crossover happens only sometimes*
- Selection and crossover together:
  - $C(S, t+1) \geq C(S, t) \cdot F(S) / F_{avg}(t) [1 - p_c \cdot d(S) / (m-1)]$

# PROOF OF TST

- Mutation:

- 1 bit will not survive:  $p_m$

- 1 bit will survive:  $1 - p_m$

- A schema will survive ( $p_m \ll 1$ ):

- $p_s(S) = (1 - p_m)^{o(S)} \longrightarrow o(S)$  is number of fixed positions

- $p_s(S) = \dots$  roughly estimate  $\dots = 1 - p_m \cdot o(S)$ , for small  $p_m$

if mutation hits \*, nothing happens

- Selection, crossover and mutation together:

- $C(S, t+1) \geq C(S, t) \cdot F(S) / F_{\text{prum}}(t) [1 - p_c \cdot d(S) / (m-1) - p_m \cdot o(S)]$

- QED.

selection  
above-average

crossover  
short

mutation  
low-order

# CONSEQUENCES OF TST AND BBH

- Encoding matters
- Size matters
- Premature convergence harms
- When GA sucks:
  - (III\*\*\*\*\*) , (\*\*\*\*\*II) are above-average
  - But  $F(III*****) \ll F(000*****00)$  – fitness function can play our opponent
  - The ideal is (IIIIIIIIII); GA has hard times finding it
  - The selection condition might be improved

# IMPLICIT PARALLELISM

- GA works with individuals, but implicitly it evolves much more schemata: something between  $2^m$  and  $n \cdot 2^m$ .
- 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.

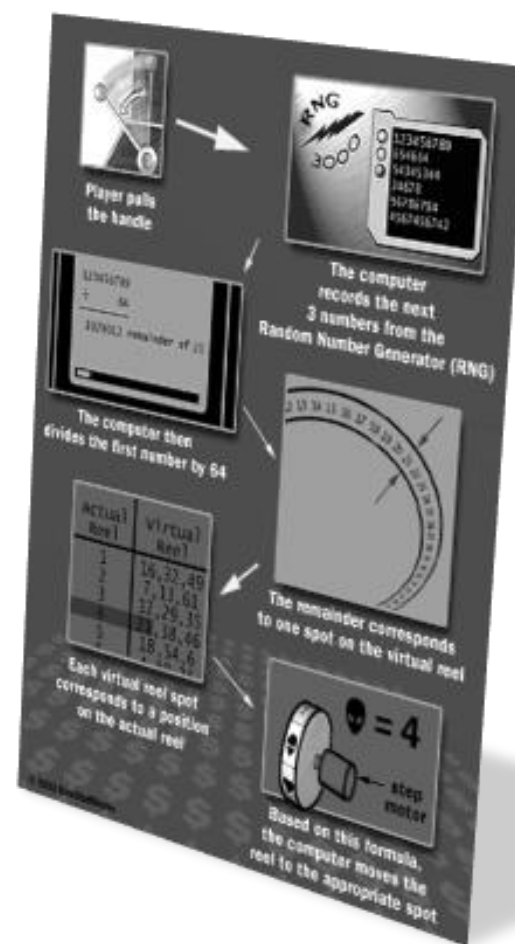
import decision about any search alg.

## EXPLORATION VS. EXPLOITATION

- Original Holland motivation:
- GA is an “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: stuck in local optima, rigidity



# I-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 minimize 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
- each arm trial costs something, but also tells me about the payoff

# 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 population
  - 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 ...

→ defined to fool SGA

- $f(x) =$ 
  - $= 2$ ; for  $x \sim 111*...*$
  - $= 1$ ; for  $x \sim 0*...*$
  - $= 0$ ; otherwise.
- For schemata we now have:
  - $F(1*...*) = 1/2$ ; → I get:  $2+0+0+0/4 = 1/2$
  - $F(0*...*) = 1$  → from definition
- But, the SGA estimates  $F(1*...*) \sim 2$ ,
- 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 pity 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 scheme  $111***...*$  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.

# TST WRAP-UP

- TST was an important first attempt to formalize GAs
- Now, more exact results exist
- The weak points of TST:
  - Populations are finite and small – the theoretical exponential increase from one generation to the next one is harmed by factors such as sampling errors, and dynamic representation of schema in populations – the longer run predictions do not hold.
  - The competition of schemata is much more complex and not independent at all.
  - TST ignores the constructive effect of operators.

# REPRESENTATION AND OPERATORS IN GA

Integer and floating point representations  
operators, selection

# ENCODING

Binary- Encoding of floating-point-domain problem is an obstacle...

- Binary
  - Classic (Holland)
  - There are nice theoretical results (better than schemata theory, we will see next semester)
  - *Holland argument: binary strings of length 100 are better than decimal of length 30 because they encode roughly the same information but have more schemata ( $2^{100} > 2^{30}$ ).*
  - But we know schemata are not that important as Holland thought
  - The important factor is that binary encoding is sometimes unnatural for a given problem.

The advice: Don't use any un-natural encodings!

Pilát a Neron, potuhli se u piva.

## OTHER ENCODINGS

- Alphabets with more symbols
- Integers
- Floating point
- Permutations,
- Trees (programs),
- Matrices,
- Neural networks (different ways),
- Finite automata
- Graphs,
- A-life agents ...

a large-number law is solving it

## SELECTION - OVERVIEW

- **Roulette-wheel selection**

- traditional, fitness-proportional

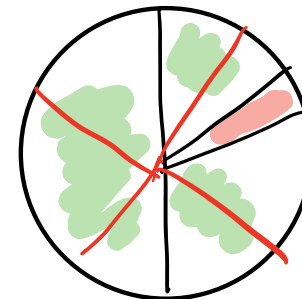
- **SUS (stochastic universal sampling)**

- Just one random position in a roulette wheel, other positions are shifts over angle  $1/n$
- „more fair roulette“ – why?

an individual having  $1/2$  should have been chosen twice more often than the one with  $1/4$ .  
This works when shifting over  $1/n$

the problem: when the population is small, some individuals are too big and take too much „selections“ / dices

this has problem with exploitation: first good solution will start to be more and more selected



$$n=4, \text{ shifts} = 1/4$$

# SELECTION - OVERVIEW

→ here, even when having one very strong individual, I give chance to the weak ones.

- **Tournament**
  - 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, multiple-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)

< > When doing mutation on an integer code of a country. Then the biased „shift“ mutation doesn't make sense, since the codes does not represent order/distance



# 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 over:
    - Typically all of them
    - Sometimes just one chosen at random
    - Sometimes a combination with 1-point crossover





*Mr. Fezziwig's Ball.*

*London, Chapman & Hall, 186, Strand.*

# A CHRISTMAS CAROL.

IN PROSE.

BEING

A Ghost Story of Christmas.

BY

CHARLES DICKENS.

WITH ILLUSTRATIONS BY JOHN LEECH.



LONDON:  
CHAPMAN & HALL, 186, STRAND.

MDCCCLXIII.



# EVOLUTION OF COOPERATION

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

# ALTRUISM VS. DARWINISM ?

- Darwinism is inherently competitive – 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 an 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)

# PRISONER'S DILEMMA

i/j	C	D
C	-1 / -1	-3 / 0
D	0 / -3	-2 / -2

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

- *Temptation > Reward > Penalty > Suckers payoff*
- *$R > P$ : mutual cooperation is better than mutual deception*
- *$T > R$  and  $P > S$ : 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
- These are called *pure strategies*

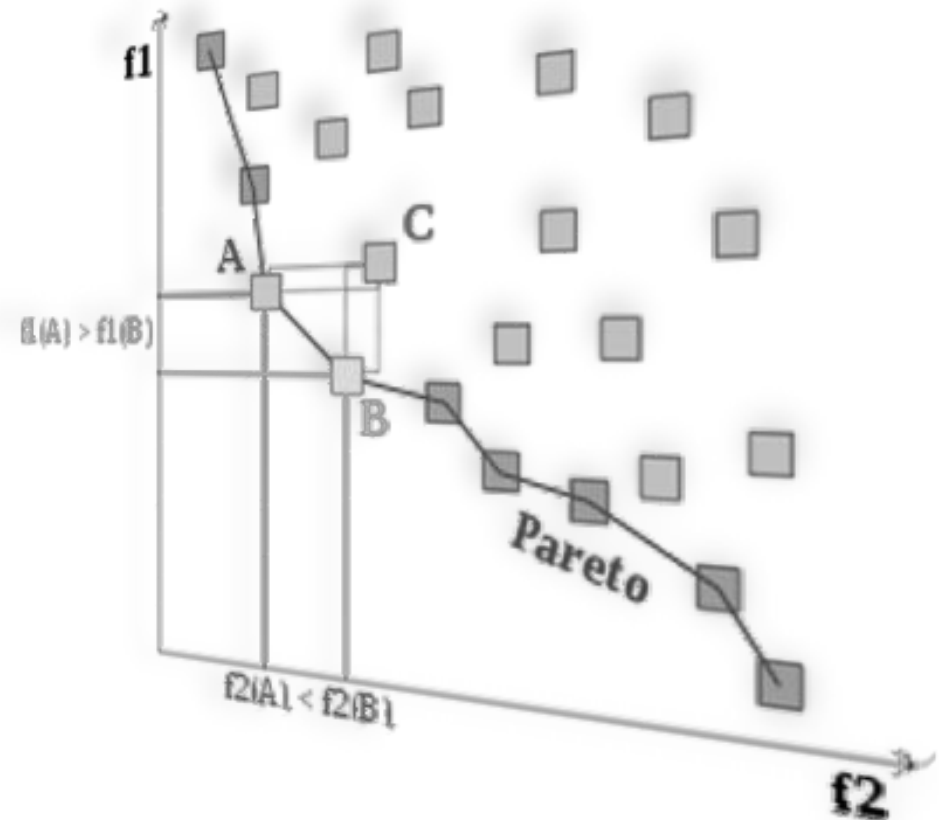
# NASH IN MIXED STRATEGIES

- *Mixed strategies* – random selection among pure strategies
- *Nash theorem*: Every game with finite number of strategies have Nash equilibrium in mixed strategies.
- Example: Rock-Paper-Scissors



# NASH AND PARETO

- 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-efficient: if an outcome of one agent can be improved without decreasing other agent's outcome



## THUS ...

- For rational agents there is no dilemma/or is there?
  - DD is Nash equilibrium
  - 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

„shadow of a future“ – we don't know how many rounds will be played.

## ITERATED PRISONER'S DILEMMA

- Players play more games, they remember the results/actions of the opponent, 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“.



→ in the last move, you can betray, it's the last.  
Then for the  $n-1$ , you can betray again, because  
you know you ~~you~~ will betray in the  $n$ -step. ...  $(n+1)$   
And you don't fear betray, because you are not punished at the end

# AXELROD TOURNAMENTS

- **The first tournament:**
  - 14 strategies plus RANDOM, 200 games, everybody played with everybody (including itself), 5x repeat
- TFT = Tit For Tat strategy — *→ as long as both are nice, you play nice.*
  - Start cooperate, then copy opponent's moves
- **The second tournament:** — *→ more new strategies, but* — *↘*
  - 62 strategies— 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?

↳ this proves that cooperation might help even for evolution

- 4 important properties of successful strategies: *— but not always,*
  - 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

*Evolutionary stable strategy: strategy wins in all populations of strategies*

# 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, consciousness, ... 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 successful
  - If the payoff  $R$  or  $P \Rightarrow C$ ,
  - if  $T$  or  $S \Rightarrow D$
- After 20 years the tournament was repeated with more strategies from each team
  - The winning strategies were cooperating as a team
  - Few moves (10) to recognize the opponent, then all strategies helped one fatter strategy from the team to get better score
  - The teams were even fighting the organizers (false teams to get more slots in the tournament ...)

→ these are the most used evolutionary solutions

# EVOLUTIONARY STRATEGIES

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

# EVOLUTIONARY STRATEGIES

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

— no any special coding  
— just the real numbers

→ we can control the evolution with it

→ always just one offspring

# ES NOTATION

- Important parameters:
  - $M$  number individuals in population
  - $L$  number of new individuals
  - $R$  number of 'parents'
- 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 – parents are killed always

+ Strategy

, Strategy

The mutation is just a way how to move in the space and sample  $f(x)$

## SIMPLEST ES EXAMPLE

having more means too much exploitations, but we want to explore mainly

- Minimize  $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- Individuals are vectors
  - $(x_1, \dots, x_n)$
- Population is just one individual
- One offspring is created
- Mutations are drawn from normal distribution
- Variance of the distribution (step size) is changed by simple heuristics to balance exploration/exploitation

Initialize  $x(0) = (x_1, \dots, x_n), t = 0$

Repeat

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

If  $f(x(t)) < f(y(t))$  then  $x(t+1) := x(t)$

else  $x(t+1) := y(t)$

Keep track of  $p$  ... % of successful mutations

Every  $k$  iterations modify  $s$  based on  $p$ :

If  $p > 1/5$  then  $s := s/c$  // explore more

If  $p < 1/5$  then  $s := s \cdot c$  // exploit more

If  $p = 1/5$  then  $s := s$  //  $0.8 \leq c \leq 1$

$t++$ , Until  $x(t)$  is good enough

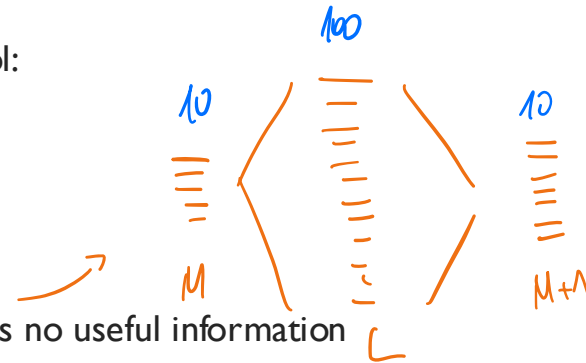
This is actually like when SGD is minimizing gradient

This is just a simple hill-climb racing ...

The bigger jump, the more we explore, otherwise exploiting

# ES SELECTIONS

- Parental selection: random – does not depend on fitness
- Environmental selection:
  - Deterministic – choose  $M$  best from the pool:
    - $(M,L)$  – the pool is just  $L$  offspring
      - Better for real-valued domains
      - Avoids local optima better
      - $M < L$ , otherwise the selection provides no useful information
    - $(M+L)$  – the pool is  $M$  parents and  $L$  offspring
      - Faster convergence
      - Recommended for discrete optimization
      - $M > L$  or  $M = L$  possible, special case  $L = 1$  is called steady-state ES



→ we can more easily stuck in local optimum  
 — because we just get less samples to choose from and also keep the previous "good ones"

# ES INDIVIDUAL

these parameters are being mutated first

→ individual newly contains some parameters to control mutation

- The individual:  $C(i) = [X_n(i), S_k(i)]$ ,  $k = 1, \dots, n$ , or  $2n$ , or  $n(n+1)/2$

- $S_k$  are **endogenous parameters** (such as standard deviations of biased floating point mutations)

- $k=1$** : One common std dev for all evolved parameters  $X$ 's

one sigma param.  
for adding  
 $+N(0, \sigma)$

- $k=n$** : Non-correlated mutations,  $n$  individual normal distributions

- Each parameter has its own std dev
- Geometrically, the mutations are within an ellipse parallel to axes

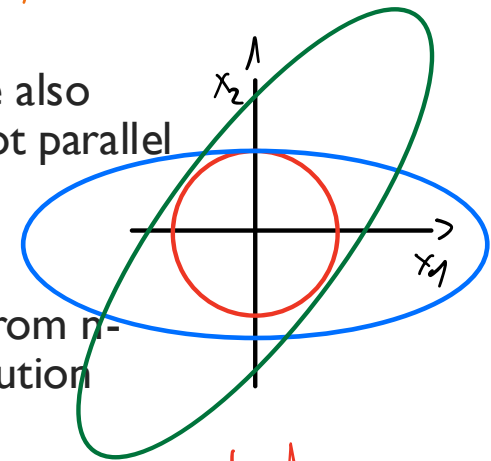
- $k=n(n+1)/2$** : Rotations are also included, the ellipses are not parallel to axes

- correlated mutations, they correspond to mutations from  $n$ -dimensional normal distribution

- Thus,  $S_k$  correspond to the covariance matrix  $C$  that is a (non-diagonal) product of rotation matrix  $M$ :  $C = M'M$

$$c_{ii} = s_i^2$$

$$c_{ij} = 1/2(s_i^2 - s_j^2) \tan(2a_{ij})$$



$k=1$

$k=n$

$$k = \frac{n \cdot (n-1)}{2}$$

reflektivita  
vztah mezi body  
- nejlepší způsob

Each part of the individual might contribute differently to the result (more / less), therefore it helps to change them separately

# ES MUTATIONS

- Endogenous parameters: - always mutate first
- 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
  - Now, multiply by a random number drawn from  $N(0,1)$
- Rotations:
  - Add a random number drawn from  $N(0,1)$
- Genetic parameters:
  - Adding random number from normal distribution with corresponding deviation, and rotation, respectively



# ES INDIVIDUAL AND MUTATION

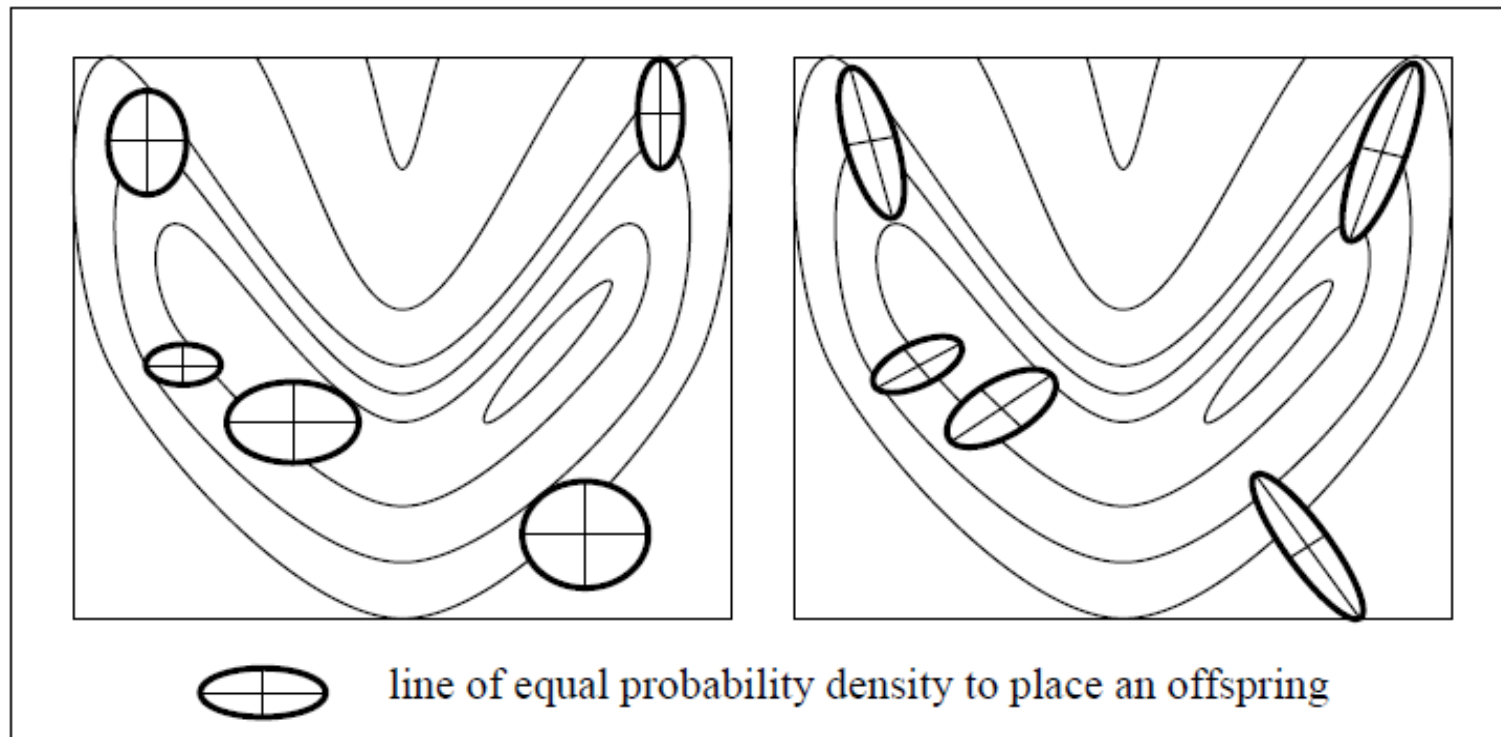


Image source: Bäck, Hoffmeister, Schwefel: A survey of Evolution Strategies (1991)

## ES CROSSOVER (CALLED RECOMBINATION HERE)

- „Gang bang“ of more parents -> one offspring
  - Local ( $R=2$ ) — traditional family
  - Global ( $R=M$ ) — gang bang
- Two versions (often combined together):
  - Uniform (dominant) — the value of a gene is selected at random from parents' genes at a particular position
  - Arithmetic (intermediate) — average of all parents' values on a particular position —> comes handy when having  $n$  parents...

# SUMMARY – ES CYCLE

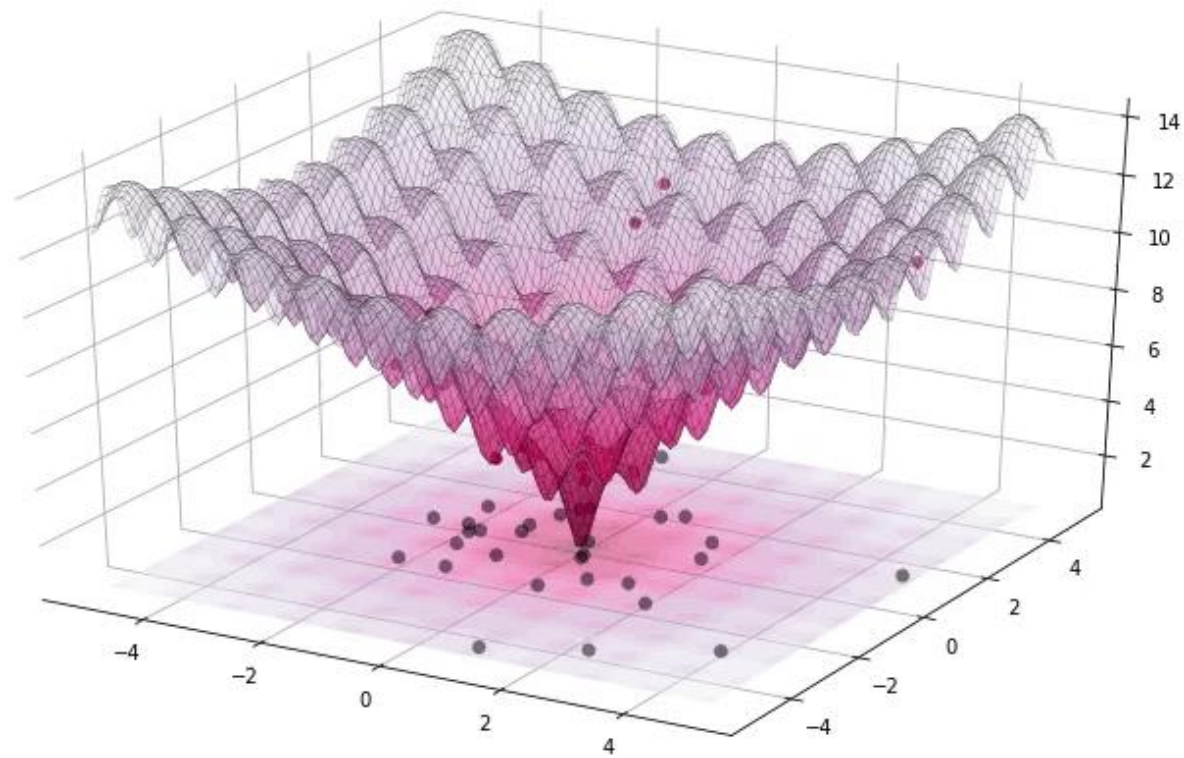
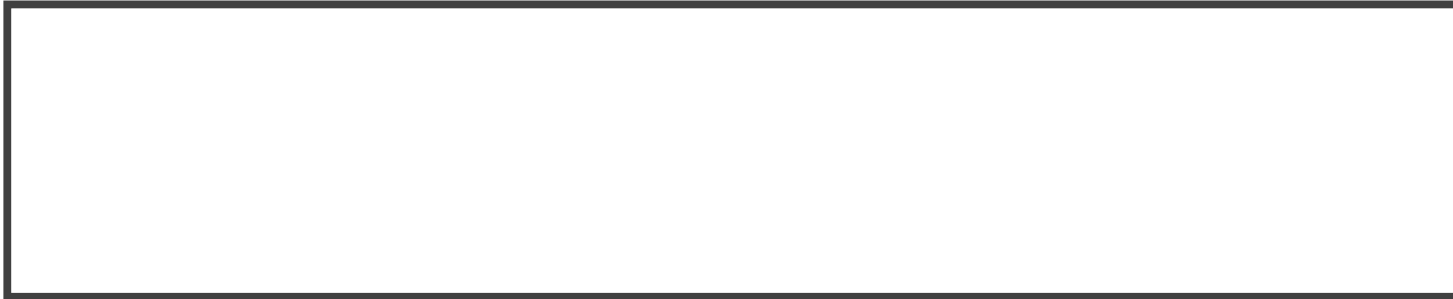
- $n=0$ ; Initialize at random a population  $P_n$  of  $M$  individuals
- Evaluate the fitness values of individuals in  $P_n$
- Until the solution is good enough:
  - Repeat  $L$  times:
    - choose  $R$  parents from  $P_n$  at random,
    - Cross them over  $\rightarrow$  one new individual  $[x,s]$ ,
    - Mutate endogenous parameters  $s$  of  $[x,s]$
    - Mutate genetic parameters  $x$  of  $[x,s]$
    - Evaluate  $[x,s]$  and put it to set of candidate solutions  $C_{n+1}$
  - For (M,L)-ES: choose  $M$  best individuals from  $C_{n+1}$  the to  $P_{n+1}$
  - For (M+L)-ES: choose  $M$  best individuals from  $P_n \cup C_{n+1}$  the to  $P_{n+1}$
- $++n$

# DIFFERENTIAL EVOLUTION

Alternative, geometrically motivated EVA

# DIFFERENTIAL EVOLUTION

- R. Storn and K. Price (1997)
- Powerful algorithm designed for black-box optimization (or derivative-free optimization).
- Find the minimum of a function  $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$ , if we do not know its analytical form.
- Relatively simple algorithm
- Used in many applications
  - European Space Agency – optimal spaceships trajectories



# DIFFERENTIAL EVOLUTION

- **Initialization:** random parameter values
- **Mutation:** „shift“ according to the others
- **Crossover:** uniform „with a safeguard“
- **Parental selection:** all individuals in a population
- **Environmental selection:** comparison and possible replacement by a better offspring

# MUTATION

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



# CROSSOVER

- Uniform crossover 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  $u_{i,p+1}$  :
- $u_{j,i,p+1} = v_{j,i,p+1}$  ; iff  $rand_{ji} \leq C$  or  $j = l_{rand}$
- $u_{j,i,p+1} = x_{j,i,p+1}$  ; iff  $rand_{ji} > C$  and  $j \neq l_{rand}$
- $rand_{ji}$  is pseudorandom number from  $<0;1>$
- $l_{rand}$  pseudorandom integer from  $<1;2; \dots ; D>$

## ENV. SELECTION

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

Mutation generates donor  $\rightarrow$  crossed with original  $\rightarrow$  if better, then saved

# MUTATION VARIANTS

*way of select / number of differences*

- The described mutation is denoted as *rand/l*
  - $X_a$  is chosen at random,  $X_b - X_c$  create one difference vector
- *rand/2*:
  - we choose  $X_b, X_c, X_d, X_e$ , and create two difference vectors:
  - $V = X_a + F(X_b - X_c + X_d - X_e)$
- *best/l*:
  - $X_a$  is not random, but the best  $X_{best}$  in population
- *best/2*: - homework

# PARTICLE SWARM OPTIMIZATION

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

# PSO

- Population-based search heuristic, Eberhart, Kennedy, 1995
- Inspiration of swarms of insect/fish
- Individual is typically a floating point vector, called a **particle**  $x \in \mathbb{R}^n$
- 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  $\leadsto \max(pBests)$

it has also  
its velocity

$(x_1 \dots x_n)$

$(v_1 \dots v_n)$

, each particle has memory of its position in history

# 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

- $\mathbf{v} := \mathbf{v} +$   
     $+ c1 * rand() * (\mathbf{pbest} - \mathbf{present}) +$   
     $+ c2 * rand() * (\mathbf{gbest} - \mathbf{present})$   
    *direction towards its best*  
    *towards the global minimum*  
    (a)  
    (b)
- $\mathbf{present} = \mathbf{present} + \mathbf{v}$
- $\mathbf{v}$  is particle speed,  $\mathbf{present}$  is particle position
- $\mathbf{pbest}$  best position of a particle in history
- $\mathbf{gbest}$  best global position in history
- $rand()$  random number from (0,1).
- $c1, c2$  constants (learning rates) often  $c1 = c2 = 2$ .

*The particles are still moving, but hopefully towards optimum*

# 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



# EVOLUTIONARY MACHINE LEARNING

Michigan vs. Pittsburg, machine learning,  
reinforcement 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 — *like simple genetic algorithm.*
      - Holland LCS: learning classifier systems
    - **Pittsburgh**: individual is a set of rules
- this is used for RL mostly, since it is more suitable.*

# CLASSIFICATION

Predicted/ Real	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Precision vs. Recall

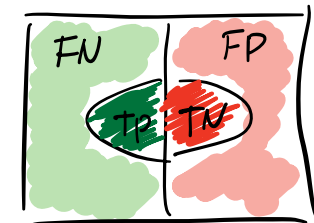
$\frac{TP}{TP + FP}$

$\frac{TP}{TP + FN}$

→ fohle virechnung  
jsem mit wicunt

- Accuracy
  - = number of correct answers / number of all answers
  - =  $(TN + TP) / (TN + TP + FN + FP)$

- Specificity
  - =  $TN / (TN + FP)$



- Sensitivity
  - =  $TP / (TP + FN)$

# MICHIGAN

Strength is actually an accuracy of all the rules given same input

- 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

2 → strength: when more rules can be applied, the strongest will apply

1101\*/11 —→ action —→ this is one rule

↪ each slot is what we expect from the feature to be (True/False)

# MICHIGAN - LCS

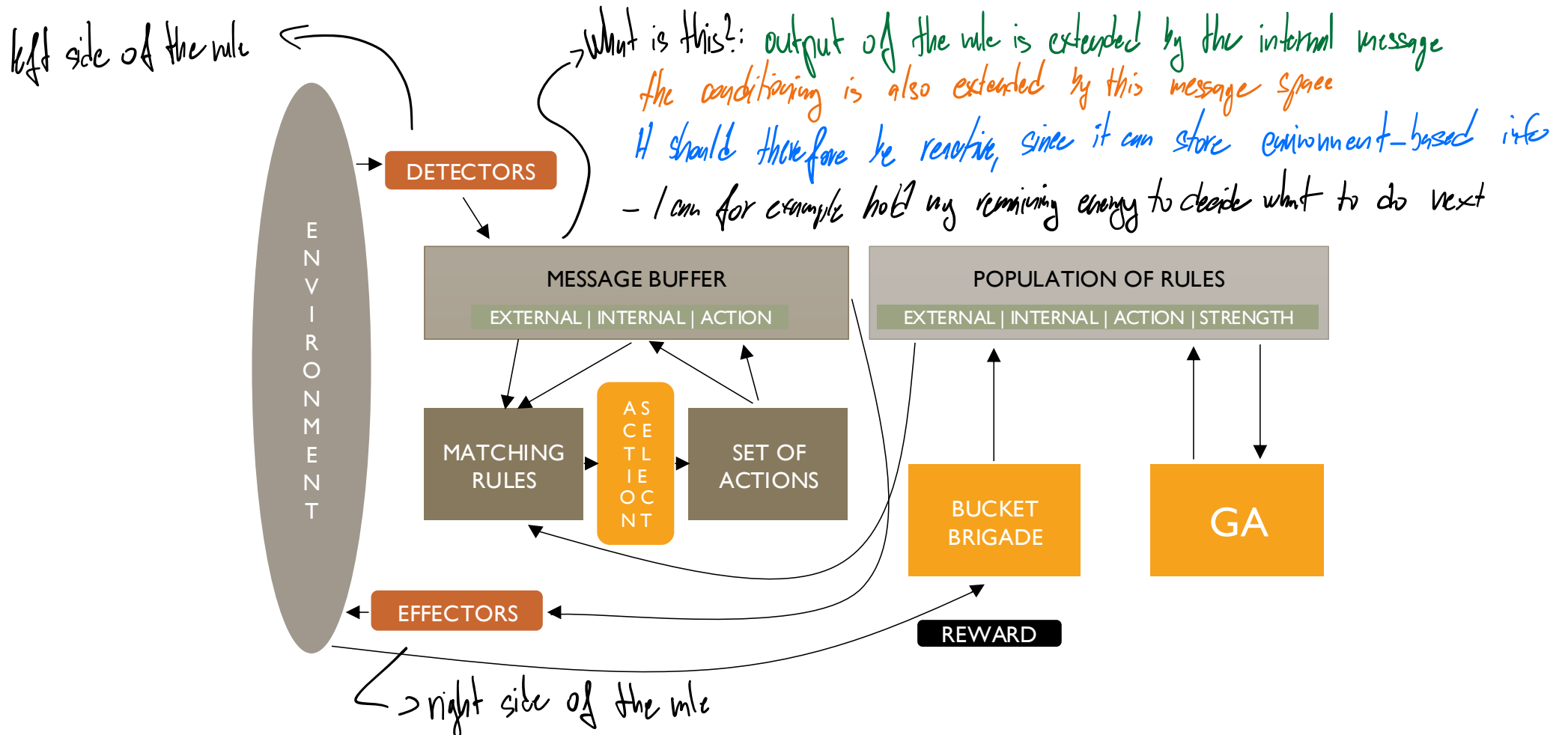
- Evolution happens only from time to time and/or on part of population
- The problem of reactivness (lack of inner memory)
  - The right-hand side of the rule contains – besides the action/classification code – other inner features, called „messages“
  - The left-hand side of the rule has special features 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

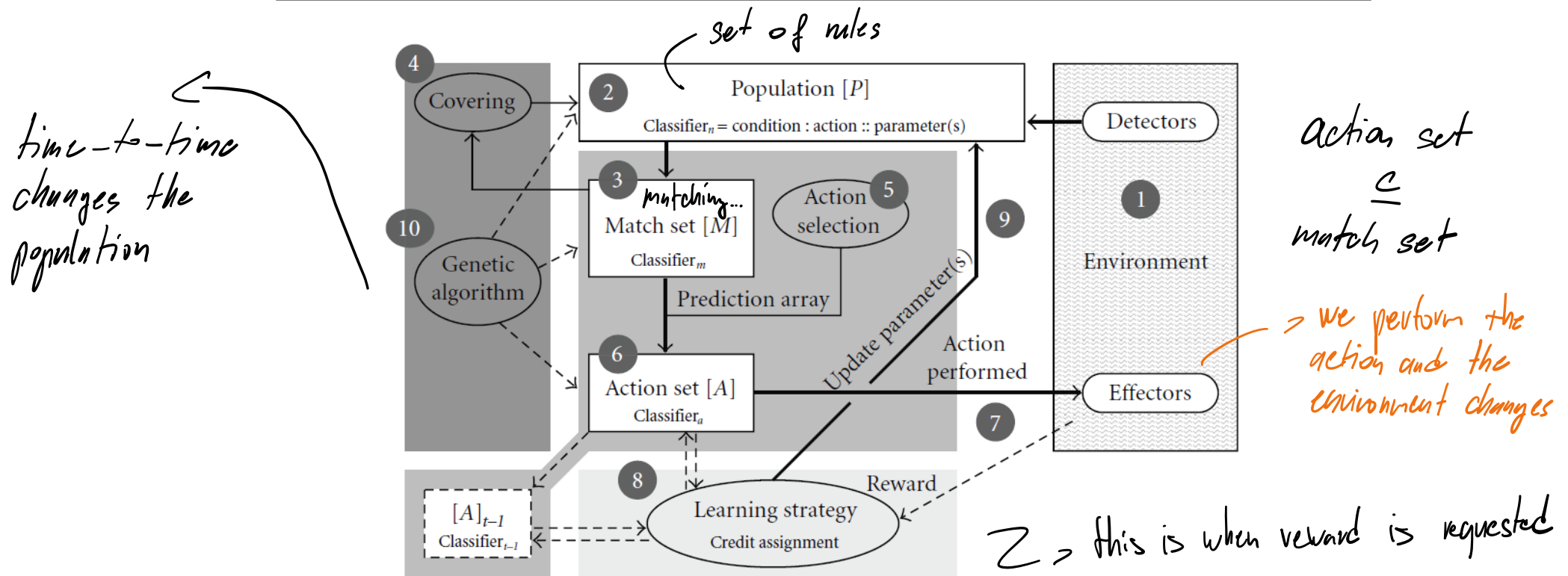
- Only some rules lead to actions that trigger reward from the environment,
- The reward should be distributed to the chain of successful rules leading to the reward
- Rules have to give up part of their strength (like paying money to take part in the action) if they compete for a chance to be applied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

↳ easy to get inflation involved

# BUCKET BRIGADE ALGORITHM



# CURRENT LCS SCHEME



- Discovery component
- Performance component
- Reinforcement component

© R. Urbanowicz, J. Moore: LCS: A complete introduction, review and roadmap, 2009



*Extended from Michigan approach*

## Z(ERO)CS

*- this is the first extended*

*- no messages, bucket brigade etc. too complex to implement well*

- (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
    - < > with each new \*, we are having higher chance of matching something*

# ZCS – ACTION SELECTION

- P – population – set of all rules
  - Detector presents an input x
  - M – match set
    - rules with condition satisfied compared to input x
  - Action a is selected from M
    - by roulette wheel mechanism
    - based on strengths of rules from M
  - A – action set – all members of M that advocated action a
- groups the same actions, sums their strengths*

# ZCS – REINFORCEMENT

essentially you remember the previous rules and reward only those that led to the current reward

- How the reward is distributed / the strength of rules is modified – operate on current  $A$  and previous  $A_{-1}$ 
  - Create a bucket  $B$  by reducing each rule from  $A$  strength by  $0 < b \leq 1$  (and adding those numbers – denote  $B$  the sum)
  - If the system receives reward  $r$  from environment after action  $a$ :
    - Add  $(b * r) / |A|$  to strength of each rule in  $A$  → rewarded winners
    - Add  $(g * B) / |A_{-1}|$  to all members of  $A_{-1}$  ( $0 < g \leq 1$ ) → rewarded helpers from previous step  
↳ inflation security
  - The strength of the rules in the set  $M - A$  (matching but advocating different action than  $a$ ) are reduced by  $0 < \text{“tax”} \leq 1$   
↳ we lower those that were giving other action than  $a$ , even though their left sides matched the input.

ZCS works well, but it has just "one-back" action rewarding. Sometimes you also need to reward the more previous actions.

## XCS – IMPROVED ZCS

Also ZCS is usually taking very general rules, but sometimes it is necessary to pick some very specialized rules, which ZCS throws away

- 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
  - The previous LCS were "strength-based" - both a measure of fitness for GA, and to control which rules "fire"
  - "Wilson's intuition was that prediction should estimate how much reward might result from a certain action, but that the evolution learning should be focused on most reliable classifiers, that is, classifiers that give a more precise (accurate) prediction" (P. Lanzi, 2000)

Rule is not as good as many times I can use it, but how well it works when used

# FITNESS - ACCURACY

## XCS - PROPERTIES

— this becomes Q-learning ♡

- Rule is a tuple (c, a, p, e, f)
  - c – condition
  - a – action
  - p – payoff prediction
  - e – prediction error
  - f – fitness used in GA
- Fitness is based on accuracy of the rule:
  - Accuracy  $k = h * (e / e_0)^{-v}$  ; h,  $e_0$  and v are parameters
- Payoff and error estimates are updated by Q-learning

# XCS – ALGORITHM SKETCH

- Based on input, form a match set  $M$ 
  - $M$  is subdivided into action sets  $A_i$  based on actions
  - We keep track of previous step action set as well
- For each action in  $A$ , predict the payoff  $p_a$  as average of payoffs of rules in the action set weighted by fitness values
- Choose the winning action  $a$  either as  $\max p_a$ , or stochastically (roulette wheel). Perform the action.

# XCS – ALGORITHM SKETCH

- Redistribute the payoff from environment to rules in the previous step Action set
  - Update accuracies using errors
  - Update fitness using accuracies
  - Update payoffs using previous step payoff and max payoff from action set
- Run GA (only) on previous Action set
  - ↳ you are providing new rules only from the good rules

# XCS – WRAPUP

- Accuracy based fitness
- Niche GA operating on action set A
- Q-learning algorithm as credit assignment
- Bridging the gap between RL and LCS
- Great applications in practice



# PITT

*Complex: how to do mutation/crossover, when I have more rules in an individual*

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

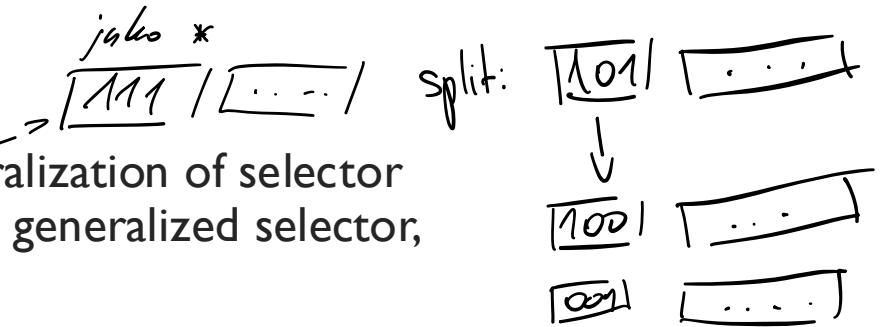
# GIL, EXAMPLE OF THE PITT APPROACH

- Binary classification tasks
- The individual classifies implicitly to one class (no right-hand side of the rules)
- Each individual is a disjunction of complexes *→ what all complexes describes him*
- Complex is conjunction of selectors (from 1 variable)  $x_1 \wedge x_2 \wedge x_3$
- Selector is a disjunction of values from the variable domain  $c_1 \vee c_2 \vee c_3$
- Representation by a bitmap:
  - $((X=A1)AND(Z=C3)) OR((X=A2)AND(Y=B2))$
  - $[001|11|0011 OR 010|10|1111]$

*We don't have right hand rules, since we are doing only classification*

# 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 1 1...1), specialization of generalized selector, inclusion of one negative example
- Operators on selectors:
  - Mutation  $0 \leftrightarrow 1$ , extension  $0 \rightarrow 1$ , reduction  $1 \rightarrow 0$ ,



# MULTI-OBJECTIVE OPTIMIZATION

Multi-Objective Evolutionary Algorithms (MOEA),  
Pareto front, NSGA II

*What can be the objectives? Going from A to B and  
optimizing both the cost and time*

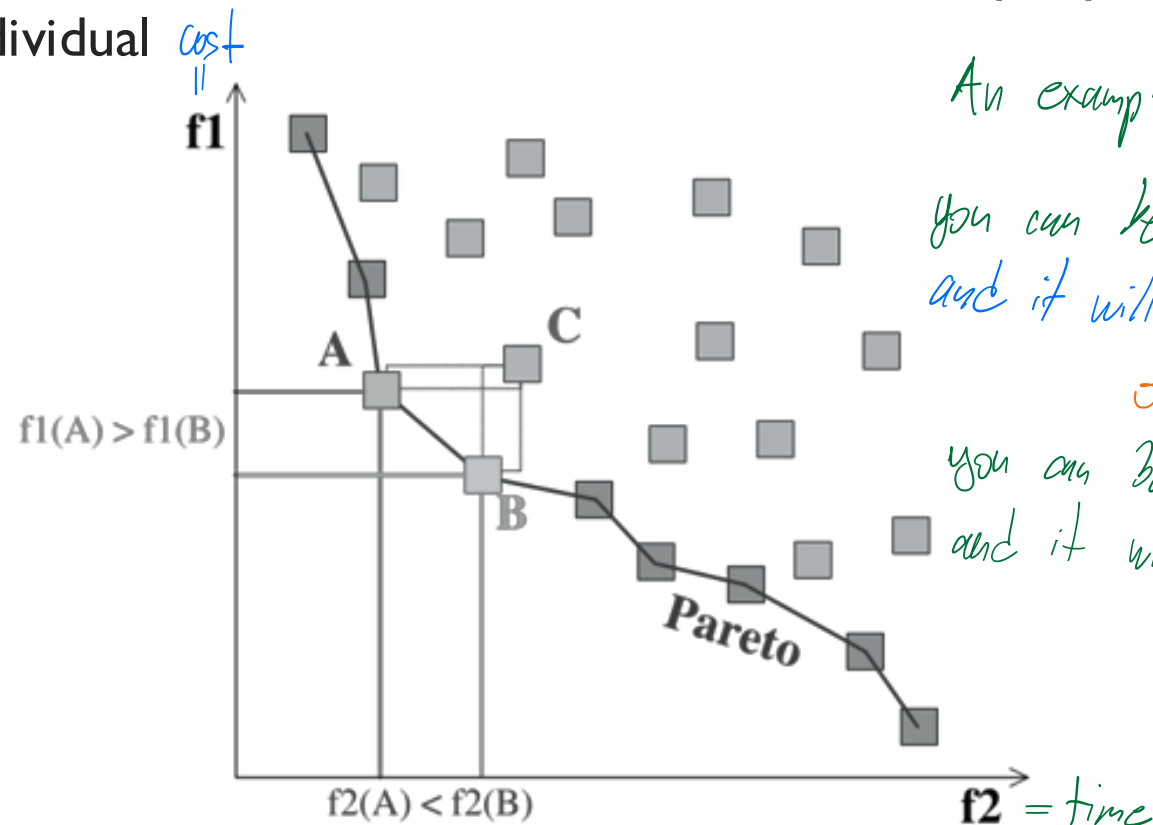
# PROBLEM

- 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  $f_i$ , which is difficult
- Definitions of dominance (of individual, or a solution):
  - Individual  $x$  **weakly dominates** individual  $y$ , iff  $f_i(x) \leq f_i(y)$ , pro  $i=1..n$
  - $x$  **dominates**  $y$ , iff it weakly dominates him, and there exists  $j: f_j(x) < f_j(y)$
  - $x$  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 individual

It gives a set of solutions, that are all optimal (can not be improved more) and user needs to choose what he wants



An example:

you can be very fast  
and it will cost a lot

or

you can be very slow  
and it will cost a little

# THE SIMPLE WAY

- How to solve MOEA in a simple (simplistic?) way:
- **Scalarization** - Aggregate the fitness:
  - Linear scalarization
    - i.e. weighted sum of all  $f_i$ , 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 it is nothing new to us, actually we were doing only SOEA so far
    - The solution of SOEA lies (somewhere) on Pareto front, but not all Pareto front points may be found
    - Nevertheless, we don't know how to set weights for  $f_i$ 's.

# THE SIMPLE WAY

- e-constraint scalarization
  - Turn MOEA into SOEA with  $n-1$  constraints
  - Choose one  $f_i$ , say  $f_1$
  - Other  $f_2 \dots f_n$  turn into constraints by choosing constants  $e_2 \dots e_n$
  - Minimize  $f_1$  with constraints  $f_2 \leq e_2 \dots f_n \leq e_n$
  - Problem how to find the constants
  - Problem how to solve the constrain optimization



# VEGA (VECTOR EVALUATED GA)

When having 3 objective functions, we take  $1/3$  of ind. with best score on  $f_0$ ,  
 $1/3$  of ind. with best score on  $f_1$   
and  $1/3$  of ind. with best score on  $f_2$ .

- 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.  $f_i$
  - 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  $f_i$

# DECOMPOSITION BASED MOEA

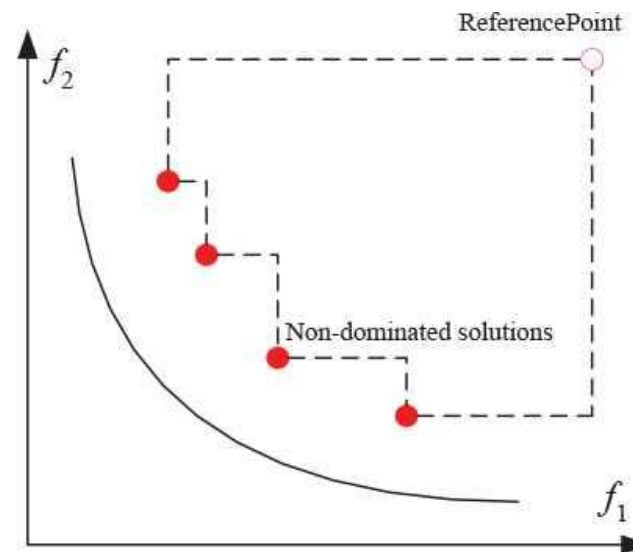
- Trying to improve basic scalarization algorithms by decomposing the population into several sub-populations scalarized with different weights
- Define several weight vectors  $L_i$  evenly distributed in the search space
- Use Chebyshev distance to perform scalarization:
  - $D(x,y) = \max_i (|x_i - y_i|)$
  - Minimize  $\max_i (L_i |f_i(x) - z_i|)$  where  $z$  is a reference point – ideal solution estimate

# MOEA/D ALGORITHM SCHEME

- input:  $\Lambda = \{\lambda(1), \dots, \lambda(\mu)\}$  {weight vectors}
- input:  $z^*$ : reference point for Chebychev distance
- initialize  $P_0 \subset X_\mu$
- initialize neighborhoods  $B(i)$  by collecting  $k$  nearest weight vectors in  $\Lambda$  for each  $\lambda(i)$
- while not terminate do
  - for all  $i \in \{1, \dots, \mu\}$  do
    - Select randomly two solutions  $x(1), x(2)$  in the neighborhood  $B(i)$ .
    - $y \leftarrow$  Recombine  $x(1), x(2)$  by a problem specific recombination operator.
    - $y \leftarrow$  Local problem specific, heuristic improvement of  $y$ , e.g. local search, based on the scalarized objective function  $g(x|\lambda(i), z^*)$ .
    - if  $g(y|\lambda(i), z^*) < g(x(i)|\lambda(i), z^*)$  then  $x(i) \leftarrow y$
    - Update  $z^*$ , if necessary, i.e. one of its component is larger than one of the corresponding components of  $f(x(i))$ . end for
  - $t \leftarrow t+1$
- return  $P_t$

# HYPER-VOLUME

- SMS-MOEA = S-metrics evolutionary multiobjective optimisation algorithms
- Find some indicator that compares two solutions, and decides which Pareto front approximation is better
- Usually a hypervolume (with some reference point) is considered as indicator, called S-metrics



# SMS-EMOA

- initialize  $P_0 \subset X_\mu$
- while not terminate do
  - {Begin variate}
  - $(x(1), x(2)) \leftarrow \text{select mates}(P_t)$  {select two parent individuals  $x(1) \in P_t$  and  $x(2) \in P_t$ }
  - $ct \leftarrow \text{recombine}(x(1), x(2))$
  - $qt \leftarrow \text{mutate}(ct)$
  - {End variate}
- {Begin selection}
  - $P_{t+1} \leftarrow \text{selectf}(P_t \cup \{qt\})$  {Select subset of size  $\mu$  with maximal hypervolume indicator}
  - from  $P \cup \{qt\}$ }
- {End selection}
- $t \leftarrow t + 1$
- end while
- return  $P_t$

# NSGA (NON-DOMINATED SORTING GA)

- 1994, an idea of dominance is used for fitness
- This still does not guarantee sufficient spread of population, it must be dealt with some other way (niching)
- Algorithm:
  - Population P is divided into consequently constructed fronts F1, F2, ...
    - F1 is a set of all non-dominated individuals from P  $\leadsto$  best approx. of p. front
    - F2 is a set of all non-dominated individuals from P-F1  $\leadsto$  2nd best approx. of p. front
    - F3 ... from P-(F1 disjunct with F2)  $\leadsto$  3rd ...
    - ... We create "pareto-like" fronts from the rest...

There are yet 3 versions from 1994, 2002 and 2014.

## NSGA CONTD.

- For each individual we compute a **niching factor**, 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 receive a dummy fitness smaller than the fitness of the worst individual from the first front, and it is again divided by their niching factor
- ... For all fronts

*how crowded is the  
surrounding of an individual.*

# NSGA II

- 2000, repairing some drawbacks of NSGA:
  - Necessity to set the right dshare value
  - Non-existence of elitism
- **Niching** → it just says "here it is crowded", "here it is not so crowded"
  - dshare – a niche count – is replaced by a **crowding distance**:  
↳ euclidian distance
  - This is a sum of distances to the nearest neighbors
  - The best individuals w.r.t. each  $f_i$ 's have crowding distance set to infinity
- **Elitism**
  - Old and new populations are joined, sorted, the better part goes to next generation

Its better to take the individual which is having less neighbours.



## 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

# NSGA III

- Improving the behavior in many dimensional problems
- Crowding distance is replaced by a set of **reference points**
  - Generate evenly distributed reference points covering the search space.
  - Consider **reference vectors** connecting 0 with reference points
  - Number of reference points = population size

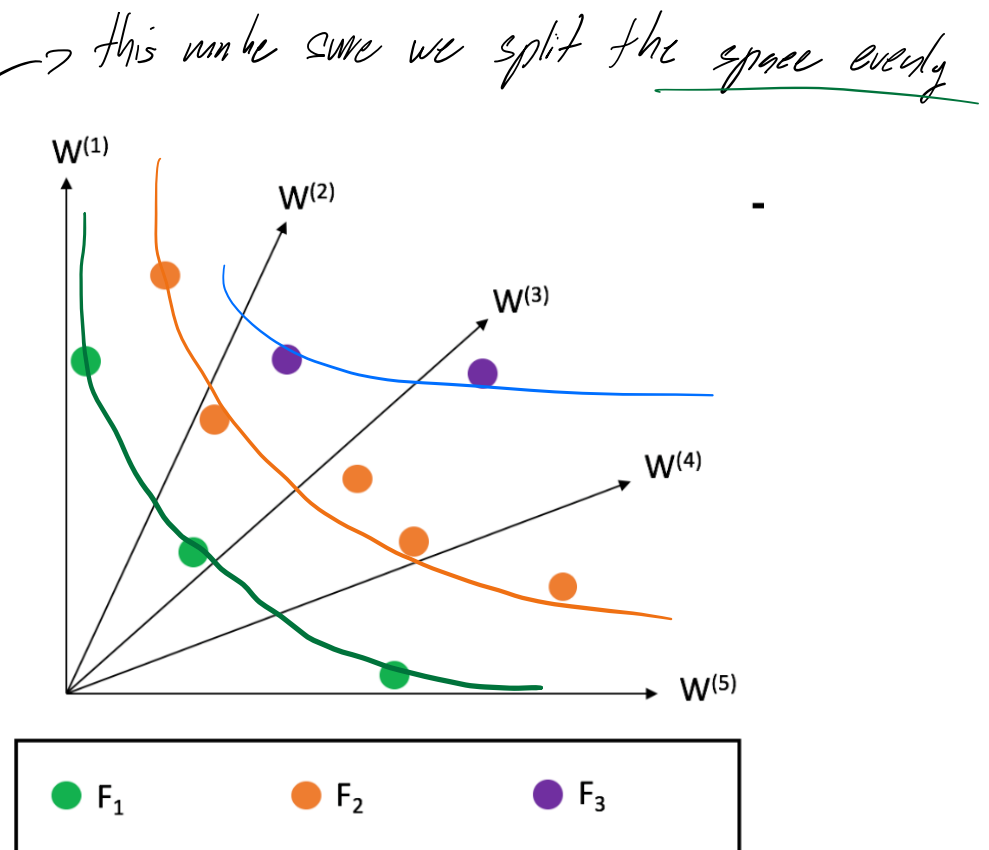


Image from <https://pymoo.org/>

# NSGA III CONTD.

## • Selection:

- first, the non-dominated sorting as in NSGA-II
- then, fill up the underrepresented reference direction first.
- If the reference direction does not have any solution assigned, then the solution with the smallest distance is surviving.
- In case a second solution for this reference line is added, it is assigned randomly.

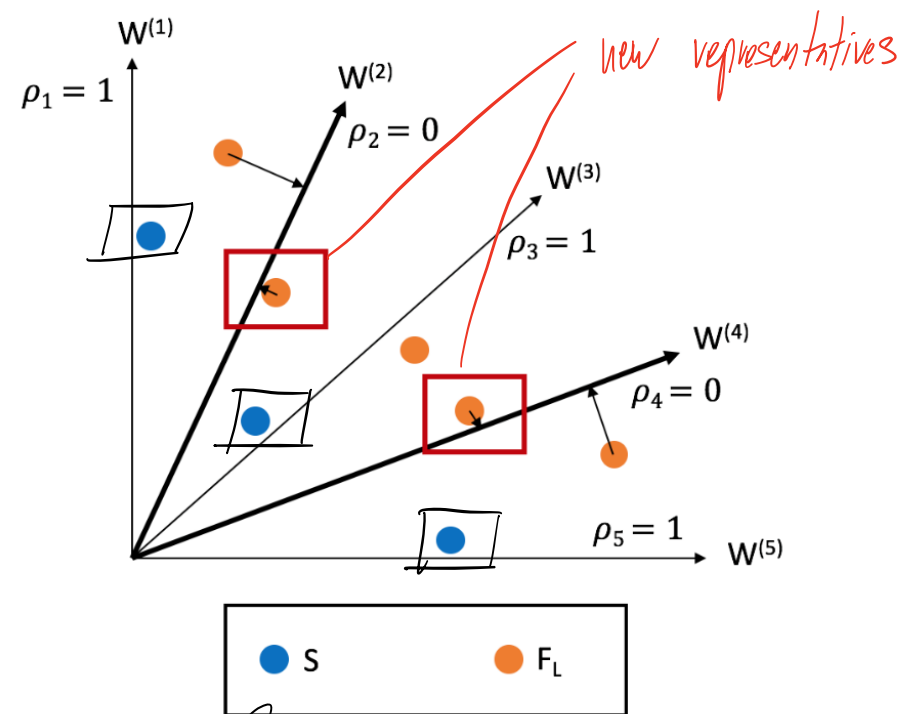


Image from <https://pymoo.org/>

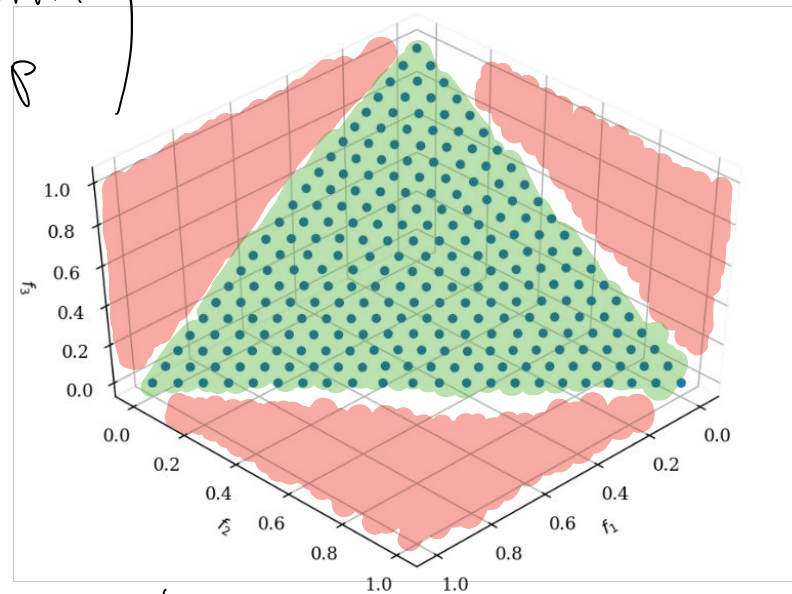
assigned to directions

$W_1$ ,  $W_3$  and  $W_5$ . Directions  $W_2$  and  $W_4$  must be assigned

# REFERENCE POINTS

- Number of points grows with dimension M.
  - $N = C(p+M+1, p)$ , where  $p$  is number of partitions, roughly equivalent to population size
- Generate evenly distributed points – not that easy
- Generate points reflecting the task preferences

$$\binom{p+M+1}{p}$$



→ not evenly distributed

→ when travelling from A to B,  
you might have some limits for time  
or distance...

# COMBINATORIAL OPTIMIZATION

EVA solves NP-hard problems, TSP, permutation representations

# EVA SOLVES HARD TASKS

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

# KNAPSACK

- Given:
  - A knapsack of capacity  $C_{MAX}$
  - $N$  items,
  - each have a price  $v(i)$
  - and a volume  $c(i)$
- The task is to choose items such that:
  - Maximize  $\sum v(i)$
  - At the same time, we squeeze them into a knapsack, i.e.
    - $\sum c(i) \leq C_{MAX}$

# 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:
  - $\max [ \sum v(i) ]$  vs.  $\min [ CMAX - \sum c(i) ]$




# KNAPSACK

- So, we have a multi-objective optimization:
  - Either weight 'em and add 'em
  - Or use your favorite MOEA from previous chapter
  - Or, change the encoding in a clever way:
    - I 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

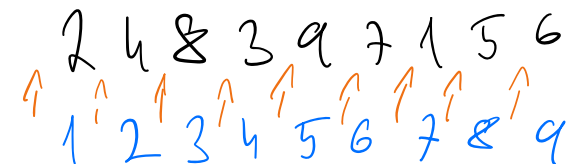
# TRAVELLING SALESMAN

- $N$  cities, tour them with minimal cost
- Fitness – the cost of the trip
- Representations 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:   $\rightarrow$  this is the order of the cities I visit
  - (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 crossover does not work
- But schemata do:
  - E.g. (\*3\*...) means all paths with 2-3 edge
- Do not use it.

was used when  
we know longer  
optimal solutions  
and wanted to  
connect them together.



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

$\hookrightarrow$  this is good for schemata!

1-2\* := there must be path from 1 to 2.

 \* 

# ORDINAL (OR BUFFER) REPRESENTATION

*"represent the cities relatively instead of exactly"*

- Motivation was to use the standard 1-point crossover

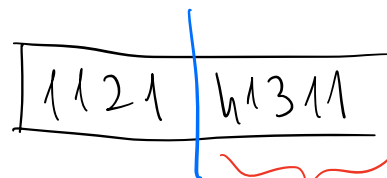
- Let us have a buffer of vertices, maybe just ordered, the encoding is in fact a position of a city in this buffer

- 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 represented as (112141311)

- Do not use it either.



*but this part of Xover was from different buffer, therefore they are very random  
But! At least they are formally correct*

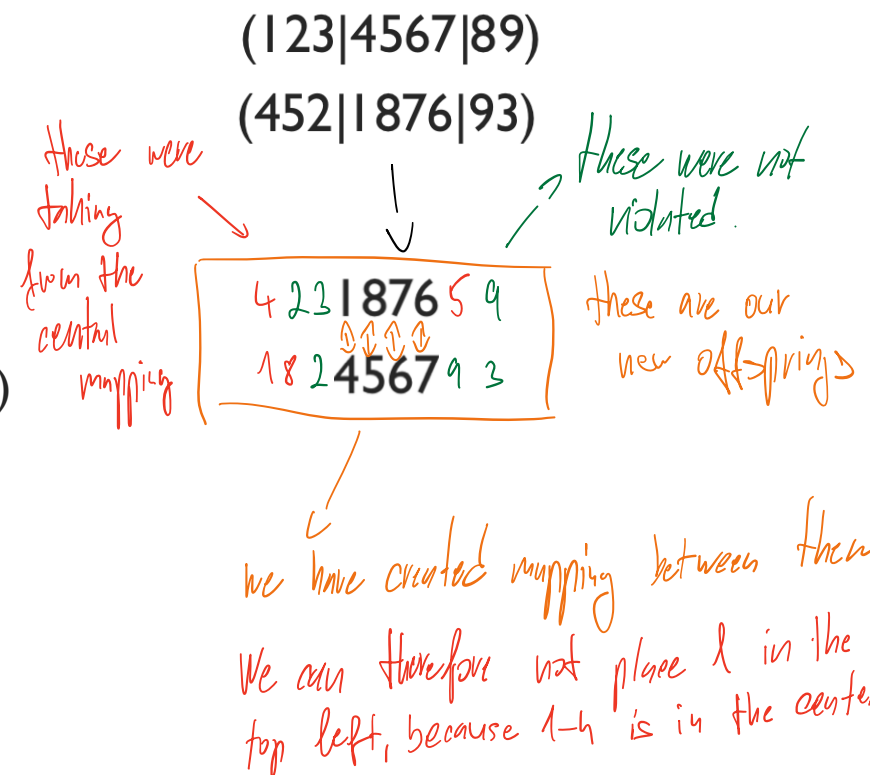
# PATH (OR PERMUTATION) REPRESENTATION

- Probably a first idea of most people
- Permutation representation is important and 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, ...

*2 - they must be valid permutation  
- they should also help the solution*

# PMX

- Partially mapped crossover (Goldberg)
- Preserve as many cities on their positions from the individuals as you can.
- 2-point
- (1 2 3 | 4 5 6 7 | 8 9) PMX (4 5 2 | 1 8 7 6 | 9 3) :
  - (... | 1 8 7 6 | ..) (... | 4 5 6 7 | ..)
  - and a mapping 1-4, 8-5, 7-6, 6-7
  - Can be added (.2 3 | 1 8 7 6 | .9) (.2 | 4 5 6 7 | 9 3)
  - According to the mapping
    - (4 2 3 | 1 8 7 6 | 5 9) (1 8 2 | 4 5 6 7 | 9 3)



# OX

*It doesn't matter where we start the trip, only the relations between cities.*

- 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

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

— >

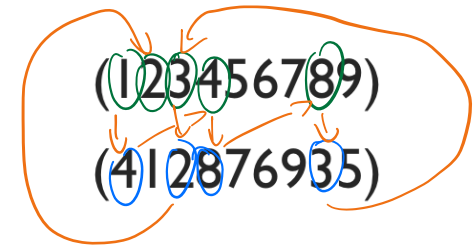
345 | ~~1~~ ~~8~~ ~~7~~ ~~6~~ | 92  
218 | 4567 | 93

- Delete crossed over cities from 1, remains: 9-3-2-1-8
- Fill the first offspring: (218|4567|93)
- Similarly, the second offspring: (345|1876|92)

*↳ we preserved the relative order between the remaining cities*

# CX

2 krasin detat vchick  
2 tech pubozeni,  
dubne k jde



- Cyclic crossover (Oliver)
- Preserve the absolute position in the path
- (1 2 3 4 5 6 7 8 9) CX (4 1 2 8 7 6 9 3 5)
  - First position at random, maybe from th first parent:  $P1=(1 \dots\dots\dots)$ ,
  - Now we have to take 4,  $P1=(1..4\dots\dots)$ , then 8, 3 a 2
  - $P1=(1 2 3 4\dots 8.)$ , can't continue, we fill from the second parent
  - $P1=(1 2 3 4 7 6 9 8 5)$
  - Similarly  $P2=(4 1 2 8 5 6 7 3 9)$



# ER

*We create an additional structure for crossover*

- 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 loses, 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% cases)

< > we will start again and choose different starting position

(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

these are common

these are common

these are common

these are common

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

lets start with 1:

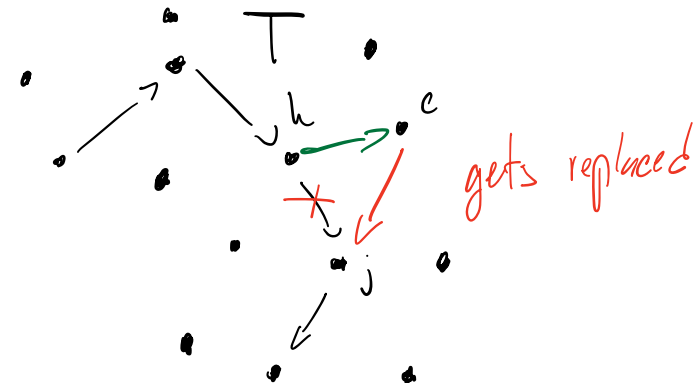
1 < 2 <sup>4</sup> - because 2 has priority, so we take it!

and only after that we would count number of edges!

# INITIALIZATION FOR TSP

→ you can even solve the problem with this, even though it would take very long.  
→ this doesn't have to be globally the shortest, but is usually better than completely random.

- Nearest neighbors:
  - Start with a random 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 (!)

1 2 3 4 5 6 7 8 9

↓

1 2 3 7 6 5 4 8 9

this is what with DNA is done as well.

- Insert a city into a path

- Shift subpath

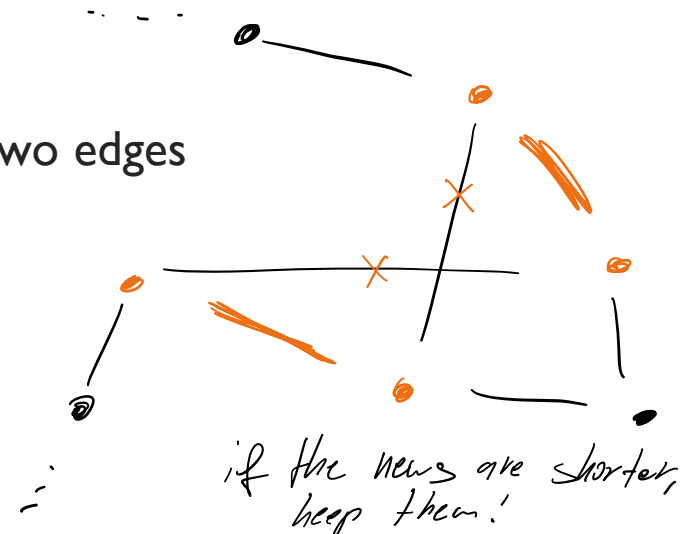
- Swap 2 cities — *that's easy...*

- Swap subpaths

- Heuristics such as 2-opt etc.

- Take two edges, four cities, choose other two edges connecting these 4 cities

*We can also go through all quadruplets of cities until finding any enhancement.*



# 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 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

# SAT

- Paradigmatic NP-complete problem of satisfiability of Boolean formula (expressed in CNF)
  - Given formula  $f: B^n \rightarrow B$  where  $B = \{0, 1\}$
  - Find evaluation  $x = (x_1, \dots, x_n)$  from  $B^n$  such that  $F(x) = 1$
- CNF:  $f(x) = c_1(x) \& c_2(x) \& \dots \& c_m(x)$ 
  - conjunction of clauses
  - each clause is disjunction of literals
  - each literal is a variable or its negation

# K-SAT

- k-SAT: each conjunction has  $k$  literals
- 2-SAT is solved in polynomial time
- 3-SAT and more are NP-complete
- Many heuristic algorithms exist for approximate SAT solving
- WSAT – popular local search heuristic evaluating solution based on number of satisfied clauses, smart selection of local search direction



# REPRESENTATIONS

- Straightforward **bit-string** - individual is Boolean vector  $x$
- **Floating-point** - encode the formula as an expression
  - Conjunction is  $*$ , disjunction is  $+$ ,
  - $x$  is  $(1-y)^2$ , non  $x$  is  $(1+y)^2$
  - Boolean 1 is 1, Boolean 0 is -1
  - Minimize the encoded formula
  - (round negative values to -1, positive to 1)

# REPRESENTATIONS II

- **Clause-based** – for each clause find feasible assignments of variables
  - the individual is a vector of assignments for all clauses
  - The length is  $m \cdot k$  for  $k$ -SAT with  $m$  clauses
  - A special fitness is needed that reflect global inconsistencies in assignments
- **Path-based** – visit clauses and select !I variable assignments in each that is consistent,
  - not all variables are assigned, the individual represents more solutions
  - Again, a special fitness solving inconsistencies is needed

# EXAMPLE

$$f(x) = (x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4)$$

$$g(y) = (y_1 - 1)^2(y_2 + 1)^2(y_4 - 1)^2 + (y_1 + 1)^2(y_3 - 1)^2(y_4 + 1)^2 + (y_2 + 1)^2(y_3 + 1)^2(y_4 - 1)^2$$

$$((x_1, x_2, x_4) = (1, 0, 0), (x_1, x_3, x_4) = (1, 1, 0), (x_2, x_3, x_4) = (0, 1, 1))$$

$$(x_1, \overline{x_4}, \overline{x_3}) \quad x = (1, 0, 0, 0) \textbf{ and } x' = (1, 1, 0, 0)$$

# FITNESS

- $f$  itself - not good
- Number of satisfied clauses  $f_{\text{MAXSAT}}(x) = c_1(x) + \dots + c_m(x),$
- Weight the problematic clauses  $f_{\text{SAW}}(x) = w_1 \cdot c_1(x) + \dots + w_m \cdot c_m(x) .$ 
  - Update after some iterations  $w_i \leftarrow w_i + 1 - c_i(x^*).$
- Refining function  $f_{\text{REF}}(x) = c_1(x) + \dots + c_m(x) + \alpha \cdot r(x)$

$$r(x) = \frac{1}{2} \left( 1 + \frac{\sum_{j=1}^n K(x_j) v_j}{1 + \sum_{j=1}^n |v_j|} \right)$$

# EVAS FOR SAT

Feature	SAWEA	RFEA	FlipGA	ASAP
replacement	$(1, \lambda^*)$	steady-state	generational	$(1 + 1)$
parent selection	–	tournament	fitness proportional	–
fitness	$f_{SAW}$	$f_{REF}$	$f_{MAXSAT}$	$f_{MAXSAT}$
initialization	random	random	random	random
crossover	–	–	uniform	–
mutation	MutOne	knowledge-based	random	random adaptive
local search	–	–	flip heuristic	flip heuristic
adaptation	fitness	fitness	–	tabu list

# OTHER TASKS - SCHEDULING

- Scheduling is NP-hard:
- Individual is a schedule, 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 constraints about the schedule quality
- Hard constraints
  - Must respect in operators, otherwise too many inadmissible solutions are generated
  - Teachers constraints, when, where what to teach, ...

# OTHER TASKS – JOB SHOP SCHEDULING

- Production planning
  - products  $o_1 \dots o_N$ , from parts  $p_1 \dots p_K$ , for each part more plans how to produce it on machines  $m_1 \dots m_M$ , machines have different times for setup to a different product
  - Fitness – production time
- Encoding is critical:
  - Permutation – plan is just a permutation 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.

# GENETIC PROGRAMMING

The very basics of tree-based representations of programs

It is hard to randomly generate legal machine code.  
Source code is not used for mutations etc.

But tree-based programming lang. is possible to mutate

many compilers use trees to compile -> therefore we can easily mutate some branches



# EVOLUTION OF PROGRAMS

- 1950s – Alan Turing proposes evolution of programs
- 1980 – Forsyth – BEAGLE: A Darwinian Approach to Pattern Recognition
- Late 1980s – Tree representations were discussed among Holland PhD students
- 1985 Michael Cramer – first description of tree individuals,
- 1989 – John Koza – tree based GP as we know it now (publication, patent)

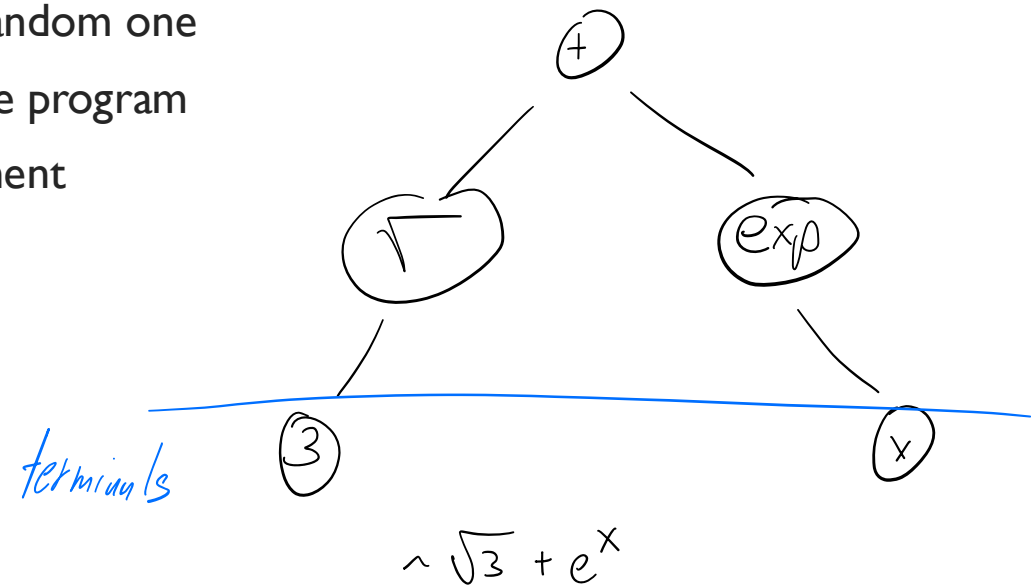
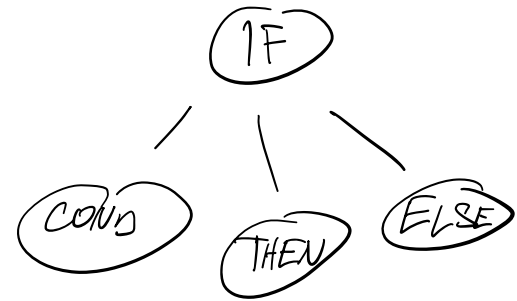
# GENERAL GP

- General structure of the GP algorithm:
  - Generate initial population of random programs
  - Evaluate the programs by running them and test on data
  - Generate new population of programs:
    - Selection based on fitness
    - Crossover of two programs
    - Mutation of programs
  - As usual, repeat until a good enough solution is found

# TREE-BASED GP

- John Koza, late 80s-early 90s:
- Programs are represented as **syntactic trees**
- **Terminals** are variables and constants
- **Non-terminals** are operations ———→ can be any other subtrees
- **Crossover** is a subtree exchange, non-terminals have typically bigger probability to be a crossover point ———→ makes more sense
- **Mutation** replaces a subtree with random one
- **Fitness** is determined by **running** the program
- **Selection** is standard, often tournament
- First examples in Lisp

mining on data  
an meaningful  
goal



# MUTATIONS

— it's best to replace subtrees instead of mutating node ~ it requires valid arity etc.

- It is good (almost necessary) to use more mutation types:
  - Random or systematic **mutation of constants**
    - GP traditionally had problems fine-tuning numerical values
    - Thus, a specialized mutations of constants speed-up the algorithm
    - Either (any) arithmetic mutation on constants
    - Or iterations of hill-climbing or other optimization methods on one or all constant set of the tree
  - Random exchange of a node for the same arity one
  - Permutations
  - Swap non-terminal for terminal
  - Mutations that decrease the size of the tree (smaller sub-tree, new individual from a sub-tree, ... )

# CROSSTOVERS

- Swap two subtrees *→ this is like one point crossover*
- Uniform crossover – GPBX (Poli, Langdon)
  - At early stages GPBX swaps large subtrees, as the population converges the operator becomes more and more local.,
  - Identify the common region C between two trees. Each node in C is considered for crossover with a constant probability.
  - Nodes in the interior of C are swapped without affecting the subtrees rooted at these nodes.
  - Nodes on the boundary of C - their subtrees are swapped.

*what is uniform  
in trees manner?*

*At the beginning: swap large subtrees (exploration), at the end: swap only little subtrees (exploitation)*

# INITIALIZATIONS

Terminals + Non terminals

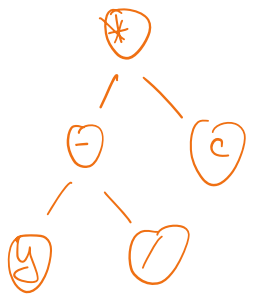
- Random procedure how to generate trees from two sets – terminals and non-terminals
- **Grow:** Generate random trees from both sets till a limit on number of nodes is reached
- **Full:** Generate random trees from non-terminal till certain depths, then only terminals are added
- **Ramped half-and-half:** half of population by grow, half by full

+ - \* /

x, y, c

↓ GROW

2kusin

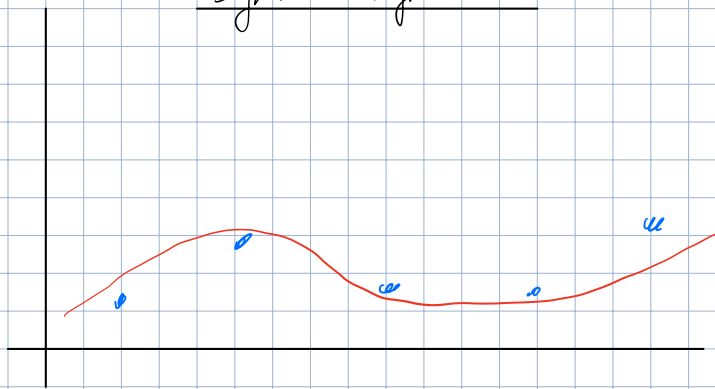


nahodně vyplňuje prázdnou, než namáznou na limit počtu prvků

Mutation is actually implemented by this same init

trying to build complete tree. First I try to use non-terminals, only at the end I use terminals

## Symbolic regression



$f(x)$

target

$$\text{error} := \sum_i (f(x_i) - t_i)^2$$

$$\text{fitness} = \text{error}$$

why symbolic?

we should provide a formula

Therefore the  $f(x)$  will be represented as trees

$T: X, C$

$NT: *, +, -, /, \sqrt{\phantom{x}}, \sin, \cos, \dots$

indiv.  $\rightarrow$  single expr. tree

fitness := error

Mutation of constants:

- this change is faster than hoping that greater subtree with non-terminals would generate the desired value.

$\hookrightarrow$

$\odot \leftrightarrow$  i need  $\rightarrow$  here

## Bloating

Because randomness says: "why not?"

- limit size
- penalize size
- smaller individuals
- shrinking mutations

I can place  $\rightarrow$  as const.

I can try to randomly build subtrees until I generate  $\rightarrow$ .

$\hookrightarrow$

$\hookrightarrow$

# WAY DOWN – LINEAR GP

- Program is represented in a linear way, most often in some machine/byte code
- Simpler, some claim more natural representation
- Simpler operators (crossover, mutation work on linear vectors)
- Faster emulation of the run
- But high risk of creating nonsense programs by mutations and crossovers
- Favourite representation in artificial life, evolution of bots and control code in games



# WAY UP – GRAPH GP

- Program is not a tree, but a more general graph, often acyclic (DAG)
- First considered as extensions of tree GP to parallel programs
- Later it was discovered, that graph structures are really useful to describe lots of things
  - Evolution of circuits
  - Finite automata, you guessed it
  - Neural networks
  - Reinforcement learning for robots, planning ...
- Complicated genetic operators – how to cross over general graphs