EVA I

NAIL 025 – 2023/24

Roman Neruda

ENGLISH VERSION – 19-12-2023

INTRODUCTION

Topics, sources, outlines.

LITERATURE

IMPORTANT

Eiben, A.E and Smith, J.E.: *Introduction to Evolutionary Computing*, (2ed) Springer, 2015.

- https://link.springer.com/book/10.1007/978-3-662-44874-8

Simon, D. : Evolutionary Optimization Algorithms, Wiley, 2013.

- https://www.wiley.com/en-gb/Evolutionary+Optimization+Algorithms-p-9781118659502

Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs* (3ed), Springer, 1996.

- https://link.springer.com/book/10.1007/978-3-662-03315-9

OF INTEREST

- 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, ...

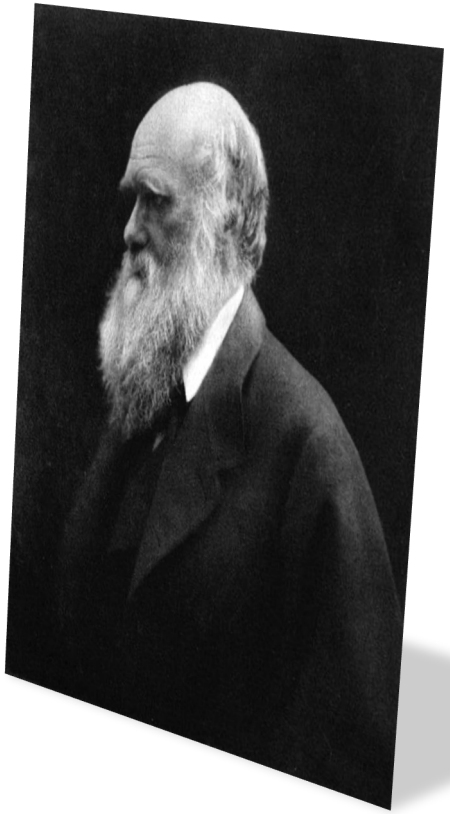
Machine learning and data mining, evolution of rule-based systems, learning classifier systems, bucket brigade algorithm, Q-learning.

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, guanin, 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->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

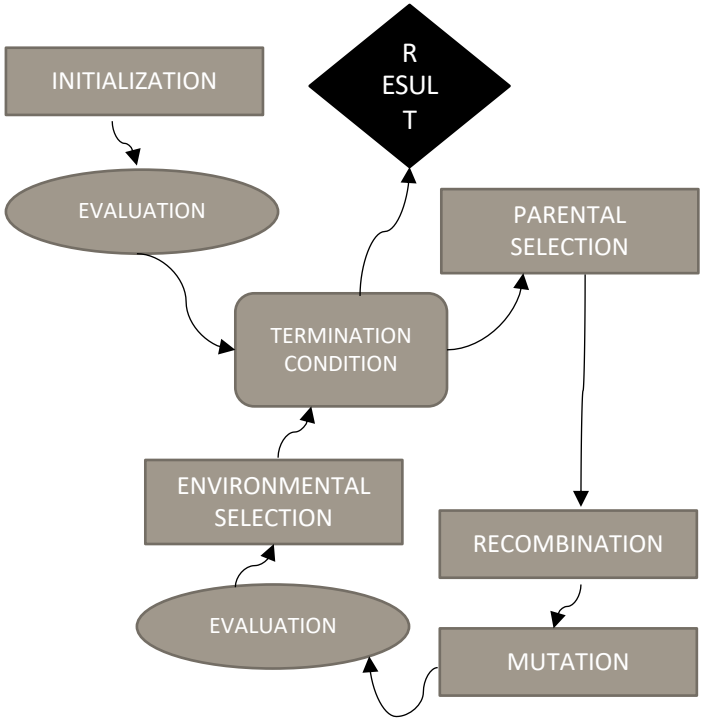
EA - SUMMARY

- 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
- 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) a 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, …)
- Evolved into several variants: linear GP, Cartesian GP, …

SIMPLE GENETIC ALGORITHM

Holland SGA, binary representation, operators and their variants

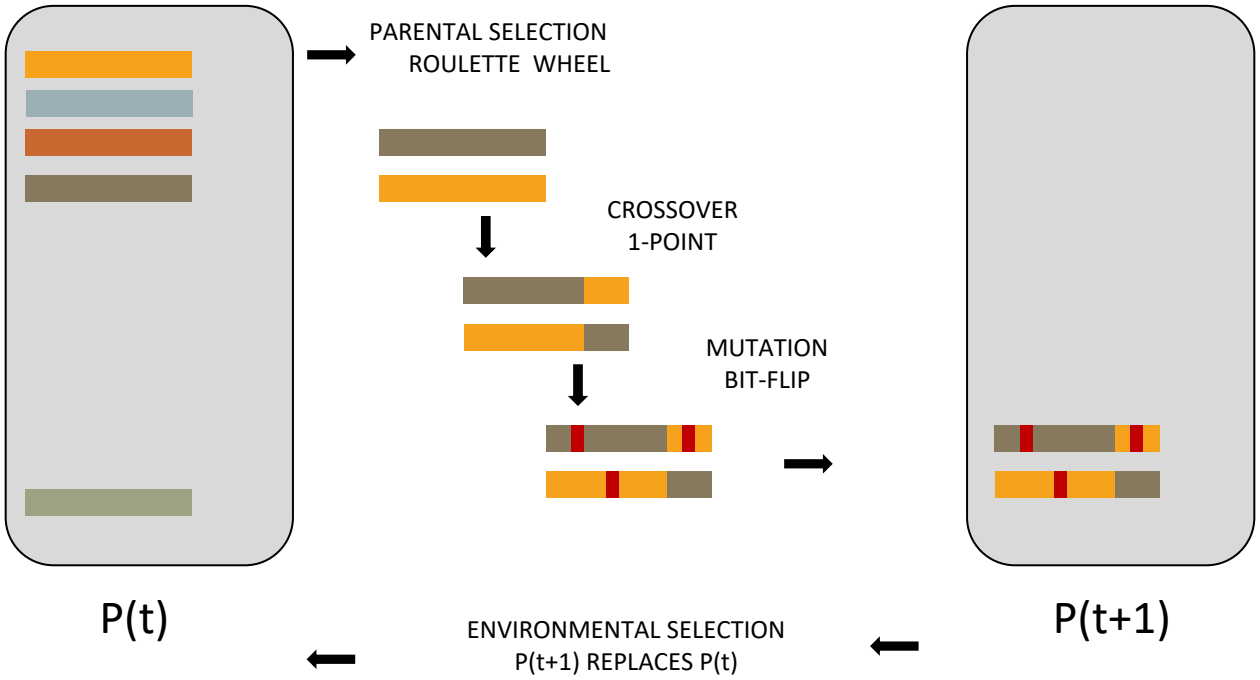
GA

- 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 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)

SGA – BASICS


PARENTAL SELECTION
ROULETTE WHEEL
CROSSOVER
1-POINT
MUTATION
BIT-FLIP
P(t)
P(t+1)
ENVIRONMENTAL SELECTION
P(t+1) REPLACES P(t)

(PARENTAL) 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

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)
- *Defining 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: $C(S,t+1)=C(S,t)\ F(S)/F_{avg}(t)$

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

PROOF OF TST

- ... Still selection:
  - So, we have: $C(S,t+1)=C(S,t)\ F(S)/F_{avg}(t)$
  - If the schema were “above-average” of e%:
  - $F(S,t)=F_{avg}(t) + e\ F_{avg}(t)$, 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:
  - $C(S,t+1) >= C(S,t) \cdot F(S)/F_{avg}(t) [1- p_c \cdot d(S) / (m-1)]$

PROOF OF TST

- Mutation:
  - 1 bit will not survive: p_m
  - 1 bit will survive: $1 – p_m$
  - A schema will survive ($p_m << 1$):
  - $p_s(S) = (1 – p_m)^{o(S)}$
  - $p_s(S) = \ldots$ roughly estimate $\ldots = 1 – p_m.o(S)$, 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)
  - The ideal is (1111111111); GA has hard times finding it
  - The selection condition might be improved

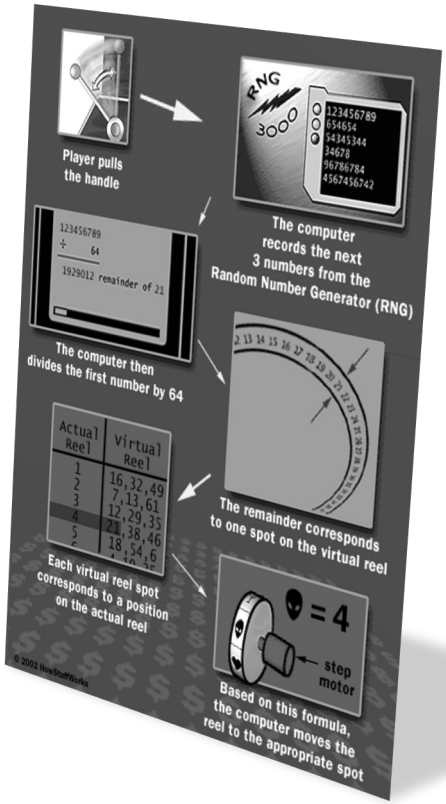
IMPLICIT PARALLELISM

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

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 ...

- f(x) =
  - = 2; for x ~ 111*... *
  - = 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 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 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.

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
  - 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 leghth 30 because they encode roughly the same information but have more schemata (2100>230).*
  - 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.

OTHER ENCODINGS

- Alphabets with more symbols
- Integers
- Floating point

- 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?

SELECTION - OVERVIEW

- **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
- **Crossove**r:
  - 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)

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

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* - *a*ffecting evolution
- New individual is accepted only if it is better
- More individuals as parents
- Todays most successful (and complex) CMA-ES (correlation matrix adaptation-ES)

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

SIMPLEST ES EXAMPLE

- Minimize f: R^n->R
- Individuals are vectors
  - $(x_1, \ldots, x_n)$
- 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) = $(x_1, \ldots, x_n)$, 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.c // exploit more

If p=1/5 then s := s // 0.8 <= c <= 1

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

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

ES INDIVIDUAL

- The individual: *C(i)=[X_n(i),S_k(i)], k=1, or 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
- **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

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

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

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

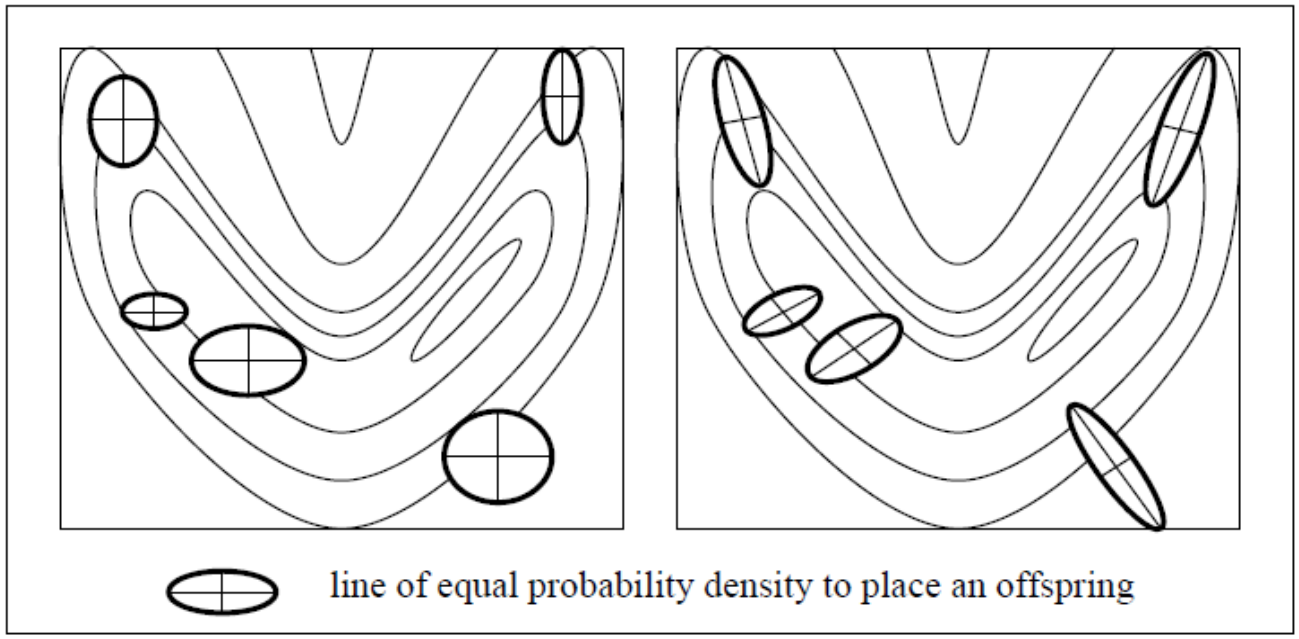


Image source: Baeck, Hoffmeister, Schwefel: A survey of Evolution Strategies (1991)

ES CROSSOVER (CALLED RECOMBINATION HERE)

- „Gang bang“ of more parents -> one offspring
  - Local (R=2)
  - Global (R=M)
- 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

SUMMARY – ES CYCLE

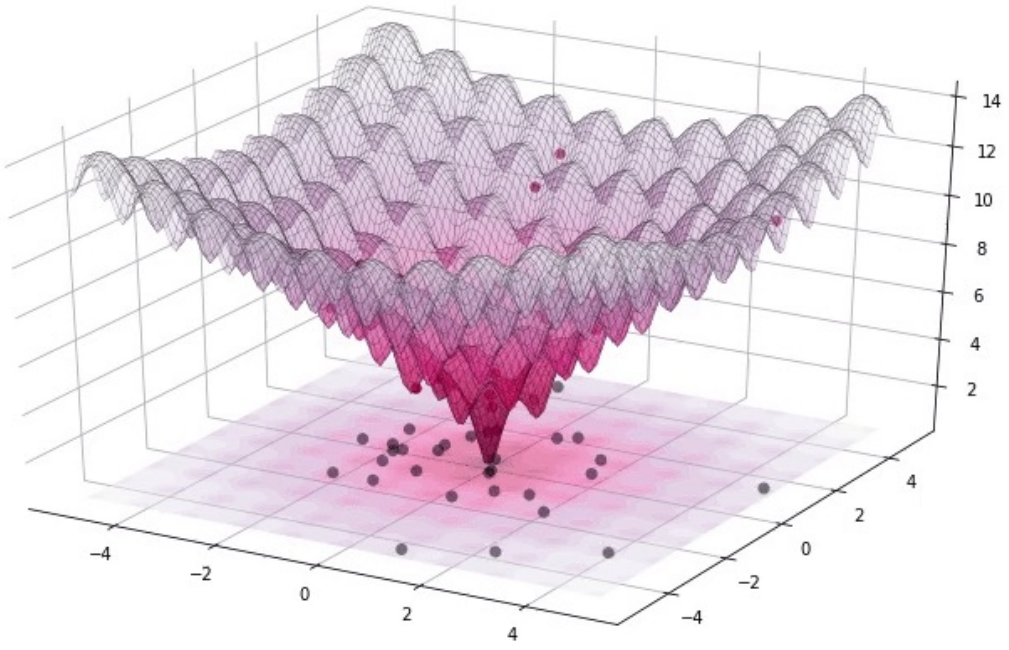
- *n=0*; Initialize at random a population P_n of *M* individuals
- Evaluate the fitness values of individuals in P_n
- Until the solution is good enough:
  - Repeat *L* times:
    - choose *R* parents from P_n at random,
    - Cross them over -> one new individual [x,s],
    - Mutate endogenous parameters 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 **U** 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)*: $R^n \rightarrow R$, if we do not know its analytical form.
- Relatively simple algorithm
- Used in many applications
  - European Space Agency – optimal spaceships trajectories



Pablo R. Mier: A tutorial on Differential Evolution with Python,
https://pablormier.github.io/2017/09/05/a-tutorial-on-differential-evolution-with-python/

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
- 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 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 $u_{i,p+1}$:
- $\boldsymbol{u}_{j,i,p+1} = \boldsymbol{v}_{j,i,p+1}$; 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>

ENV. 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 $i=1,2, \dots , N$
- Mutation, crossover, and selection is repeated until some termination criterion is satisfied (typically, the fitness of the best individual is good enough)

MUTATION VARIANTS

- The described mutation is denoted as *rand/1*
  - Xa is chosen at random, Xb – Xc create **one** difference vector
- *rand/2:*
  - we choose Xb, Xc, Xd, Xe, and create **two** difference vectors:
  - V = Xa + F(Xb - Xc + Xd –Xe)
- *best/1:*
  - Xa is not random, but the best Xbest 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 – boids, Reynolds, 1986
- Individual is typically a floating point vector, 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*** *=* ***present*** *+* ***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

MULTI-OBJECTIVE OPTIMIZATION

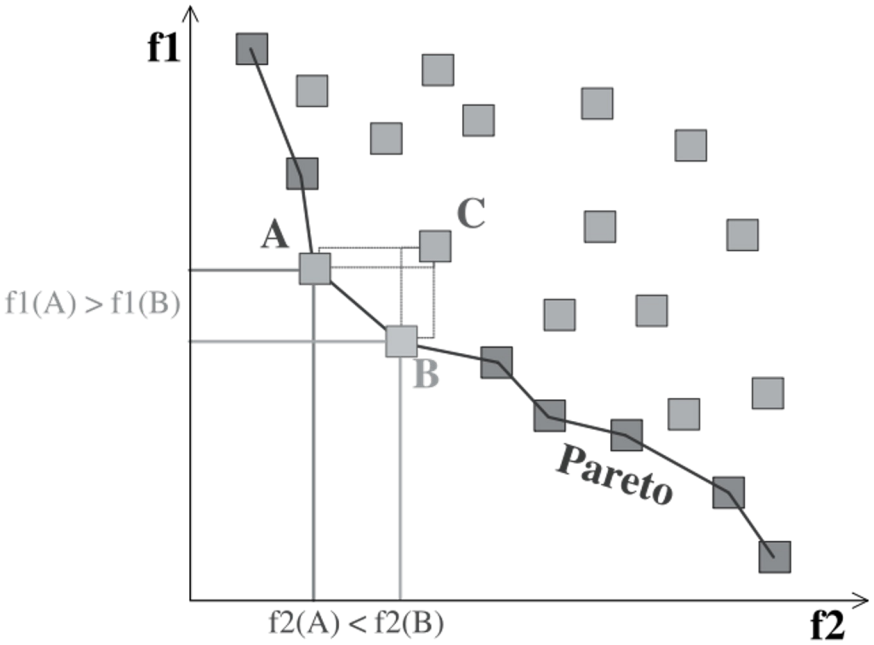
Multi-Objective Evolutionary Algorithms (MOEA),
Pareto front, NSGA II

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 individual



THE SIMPLE WAY

- How to solve MOEA in a simple (simplistic?) way:
- **Scalarization** - Aggregate the fitness:
  - Linear scalarization
    - 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
    - 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 *fi*'s.

THE SIMPLE WAY

- e-constraint scalarization
  - Turn MOEA into SOEA with n-1 constraints
  - Choose one fi, say f1
  - Other f2 ... fn turn into constrains by choosing constants e2 … en
  - Minimize f1 with constraints f2<e2 … fn<en
  - Problem how to find the constants
  - Problem how to solve the constrain optimization

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*

DECOMPOSITION BASED MOEA

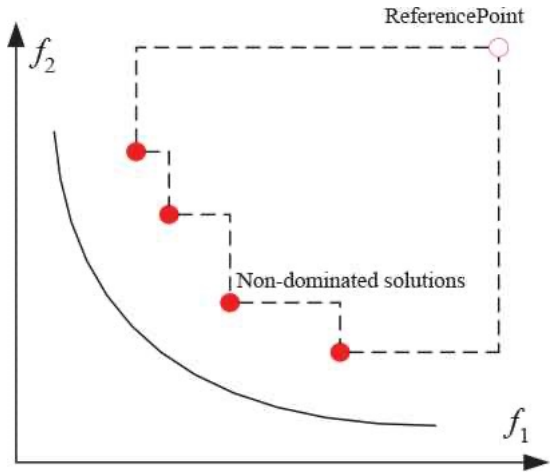
- Trying to improve basic scalarization algorithms by decomposing the population into several sub-populations scalarized with different weights
- Define several weight vectors L_i evenly distributed in the search space
- Use Chebyshev distance to perform scalarization:
  - $D(x,y) = \max_i (|x_i - y_i|)$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

MOEA/D ALGORITHM SCHEME

- input: $\Lambda = \{\lambda(1),...,\lambda(\mu)\}$ {weight vectors}
- input: $z*$: reference point for Chebychev distance
- initialize $P0 \subset X\ \mu$
- initialize neighborhoods B(i) by collecting k nearest weight vectors in Λ for each $\lambda(i)$
- while not terminate do
  - for all $i \in \{1,...,\mu\}$ do
    - Select randomly two solutions x(1), x(2) in the neighborhood B(i).
    - y ← Recombine x(1), x(2) by a problem specific recombination operator.
    - y ← 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) ← y
    - Update $z*$, if neccessary, i.e, one of its component is larger than one of the corresponding components of f(x(i)). end for
  - t←t+1
- return Pt

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 P0 ⊂ Xμ
- while not terminate do
  - {Begin variate}
  - (x(1), x(2)) ← select mates(Pt) {select two parent individuals x(1) ∈ Pt and x(2) ∈ Pt}
  - ct ← recombine(x(1), x(2))
  - qt ← mutate(ct)
- {End variate}
- {Begin selection}
  - Pt+1 ← selectf (Pt ∪ {qt}) {Select subset of size μ with maximal hypervolume indicator
  - from P ∪ {qt}}
- {End selection}
- t ← t + 1
- end while
- return Pt

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
    - 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 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 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 neighbors
  - The best individuals w.r.t. each *fi's* have crowding distance set to infinity
- **Elitism**
  - 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

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

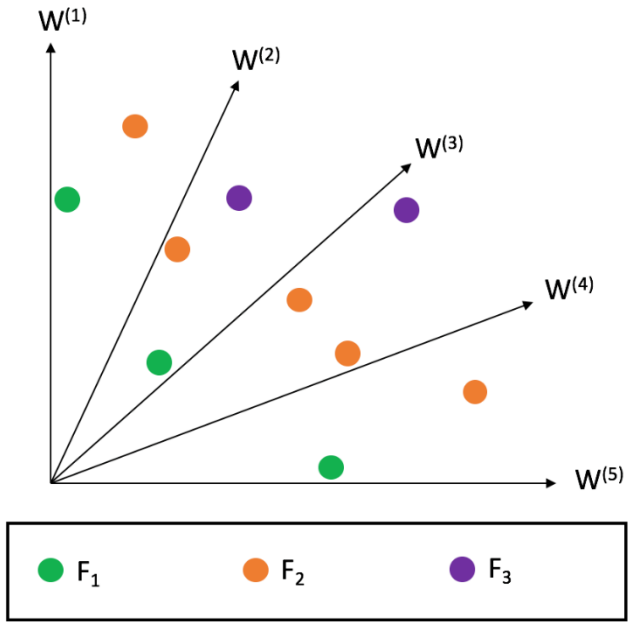


Image from https://pymoo.org/

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.

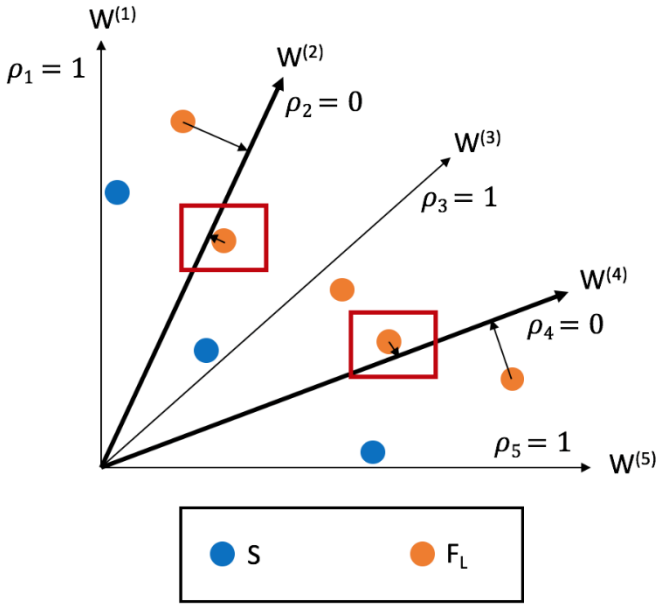
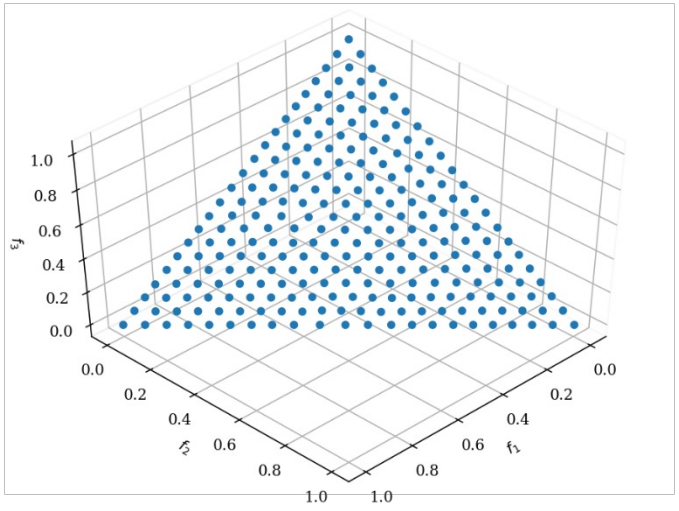


Image from https://pymoo.org/

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



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
    - Holland LCS: learning classifier systems
  - **Pittsburgh**: individual is a set of rules

CLASSIFICATION

Predicted/Real	Positive	Negative
Positive	TP	FP
Negative	FN	TN

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

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

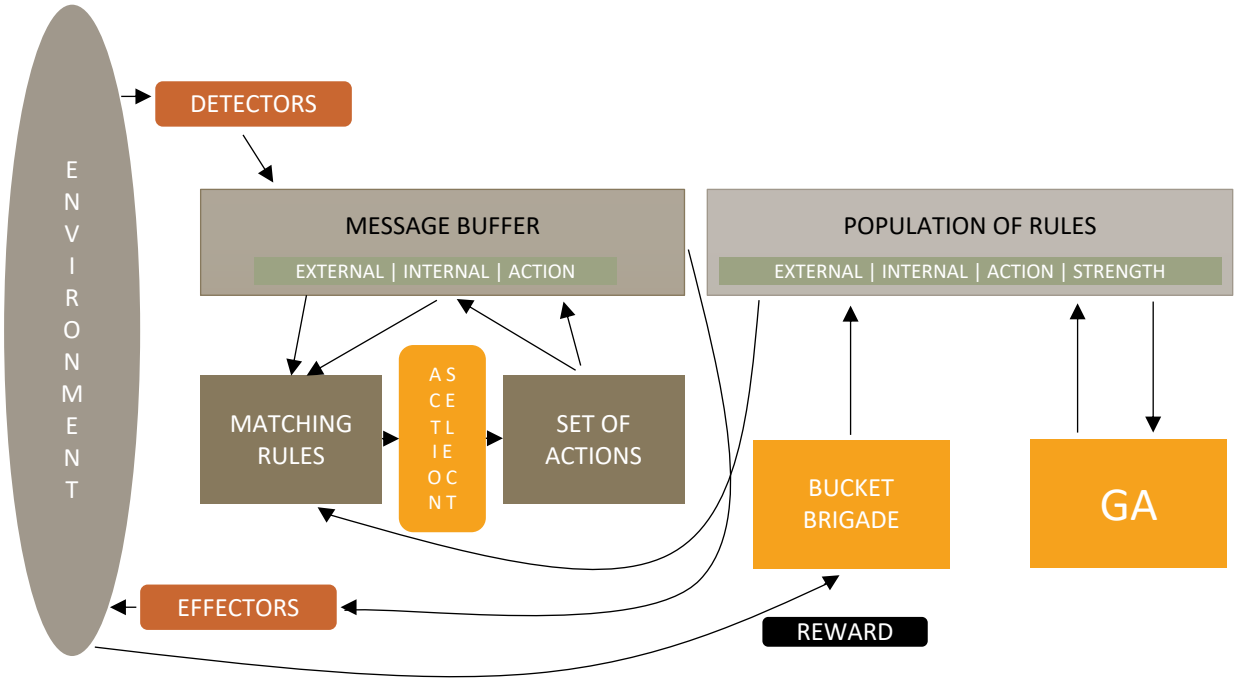
MICHIGAN - LCS

- 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

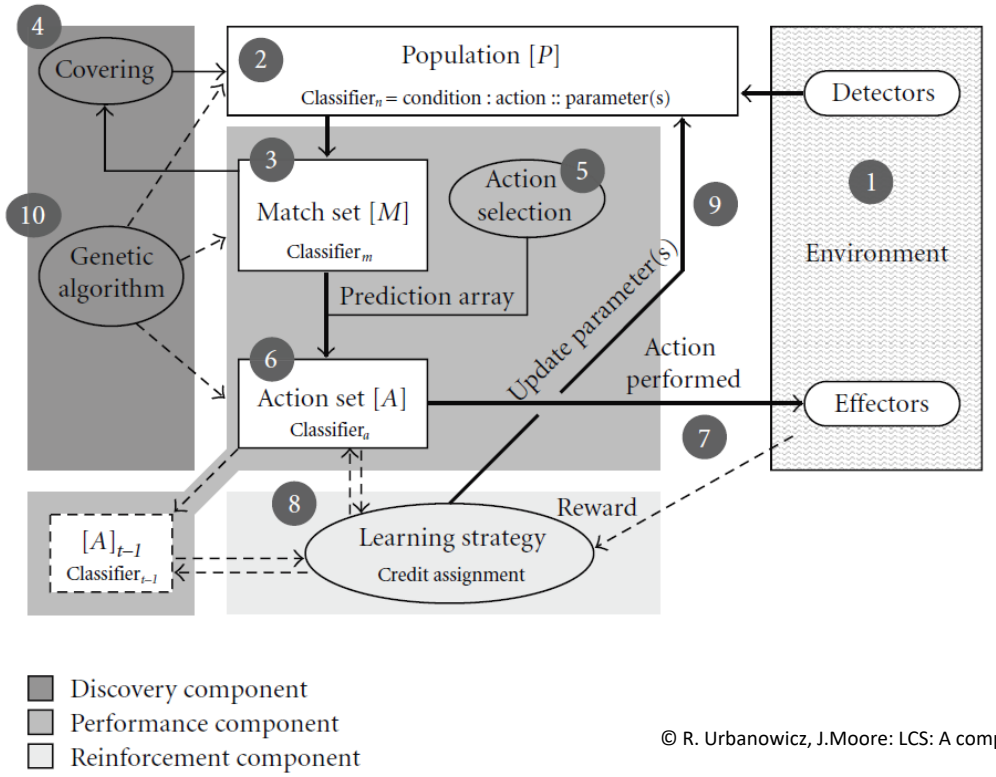
LCS – BUCKET BRIGADE

- 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 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

BUCKET BRIGADE ALGORITHM



CURRENT LCS SCHEME



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 – 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

ZCS – REINFORCEMENT

- 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 $0<b<=1$ (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
    - Add (g*B) / $|A_{-1}|$ to all members of A_{-1} $(0<g<=1)$
  - The strength of the rules in the set M – A (matching but advocating different action than a are reduced by 0<“tax”<=1

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
  - 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)

XCS - PROPERTIES

- 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 Ai 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

XCS DICTIONARY (HTTPS://PYTHONHOSTED.ORG/XCS/)

- **Classifier rule** (*rule* or a *classifier)*
  - pairing between a condition, describing which situations can be matched, and a suggested action.
  - **prediction** indicating the expected reward if the suggested action is taken when the condition matches the situation,
  - **fitness** indicating its suitability for reproduction and continued use in the population
  - **numerosity** value which indicates the number of (virtual) instances of the rule in the population.

XCS DICTIONARY (HTTPS://PYTHONHOSTED.ORG/XCS/)

- **Classifier set** (*population)*
  - collection of all rules currently used and tracked by the classifier.
  - GA operates on this set of rules over time to optimize them for accuracy and generality in their descriptiveness of the problem space.
  - population is virtual, if the same rule has multiple copies in the population, it is represented only once, with an associated numerosity value

XCS DICTIONARY (HTTPS://PYTHONHOSTED.ORG/XCS/)

- **Match Set**
  - set of rules which match against the current situation.
- **Action Set**
  - set of rules which match against the current situation and recommend the selected action.
  - action set is a subset of the match set.
  - match set can be seen as a collection of mutually exclusive and competing action sets, from which only one is to be selected.

XCS DICTIONARY (HTTPS://PYTHONHOSTED.ORG/XCS/)

- **Reward**
  - value which acts as the signal the algorithm attempts to maximize
  - *Immediate reward* (*raw reward*) is the original, unaltered reward value returned by the scenario in response to each action.
  - *Expected future reward* is the estimated payoff for later reward cycles, specifically excluding the current one; the prediction of the action set on the next reward cycle acts in this role in the canonical XCS algorithm.
  - *Payoff* or *combined reward* is the combined sum of the immediate reward, plus the discounted expected future reward.

XCS DICTIONARY (HTTPS://PYTHONHOSTED.ORG/XCS/)

- **Prediction**
  - estimate by a classifier rule or an action set as to the payoff expected to be received by taking the suggested action in the given situation.
  - prediction of an action set is formed by taking the fitness-weighted average of the predictions made by the individual rules within it.
- **Fitness**
  - Each rule has associated fitness value used as a guide for selection
  - In XCS, as opposed to strength-based LCS variants such as ZCS, the fitness is actually based on the accuracy of each rule's reward prediction, as opposed to reward size.
  - rule with a very low expected reward can have a high fitness provided it is accurate in its prediction of low reward.
  - Using reward prediction accuracy instead of reward prediction size helps XCS find rules that describe the problem in a stable, predictable way.

XCS – WRAPUP

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

PITT

- 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

- Janikow, 1993
- Binary classification - 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,

GASSIST, ANOTHER EXAMPLE OF THE PITT APPROACH

- Bacradit, 2004
- Goals:
  - Improving the generalization capacity of the model,
  - Reducing the run-time of the system ,
  - Proposing representations for real-valued attributes.
- Default rule
  - Decide which class is “default”
  - A special rule IF anything THEN default
  - Not influenced by operators

GASSIST, ANOTHER EXAMPLE OF THE PITT APPROACH

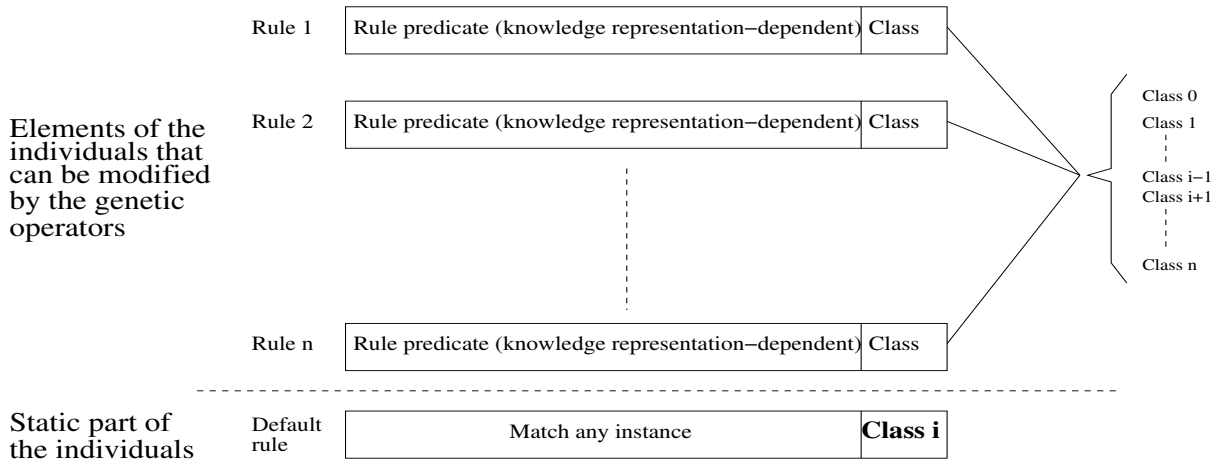


Image from Bacradit, 2004

GASSIST, ANOTHER EXAMPLE OF THE PITT APPROACH

- Adaptive discretization intervals rule representation
  - A set of static discretization intervals (called *micro-intervals*) is assigned to each attribute term of each rule of each individual, they can be on/off (0 or 1)
  - Intervals of the rule are built joining together adjacent micro-intervals.
  - Mutation will affect the semantic part of the rule: the states (0 or 1) of each interval in the rule.
  - Split an interval - select a random point in its micro-intervals to break it.
  - Merge two intervals - the state (1 or 0) of the resulting interval is taken from the one which has more micro-intervals. 8. The discretization assigned in the initialization stage to each attribute term is chosen from a predefined set.
  - Number and size of the initial intervals is selected randomly.
  - Cut points of the crossover operator can only take place in attribute terms boundaries, not between intervals.

GASSIST, ANOTHER EXAMPLE OF THE PITT APPROACH

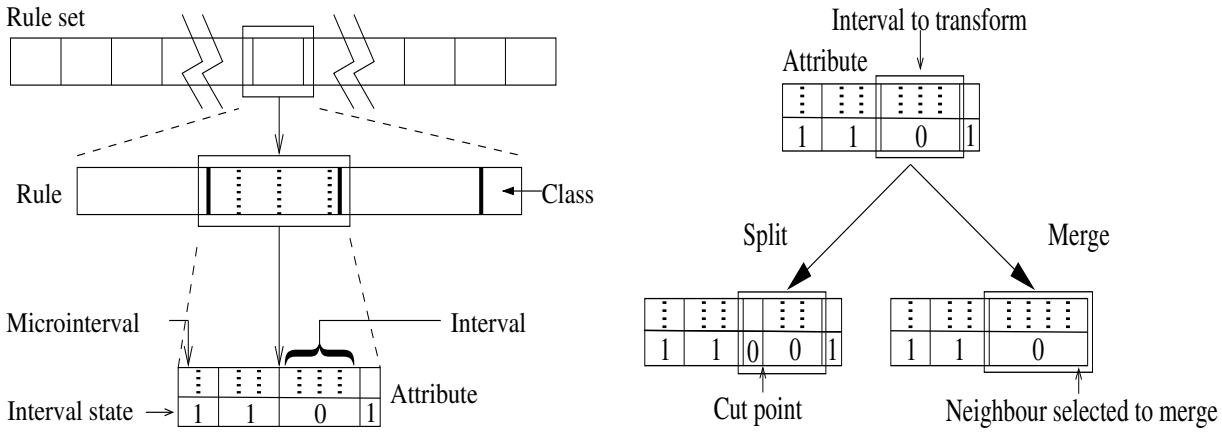


Image from Bacradit, 2004

PITT WRAP-UP

- Individual is complete rule-based system
- (Much) more used for classifications
- Rich domain representation of conditions
- Rich (too much?) repertoire of operators

- Common use of fuzzy sets – evolve fuzzy rules

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 CMAX
  - 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.
    - $\sum c(i) <= CMAX$

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 [∑v(i)] vs. min [CMAX – ∑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:
    - 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 – 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:
  - (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.

ORDINAL (OR BUFFER) REPRESENTATION

- 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.

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, ...

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 added (.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 offspring: (218|4567|93)
  - Similarly, the second offspring: (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% cases)

(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 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 (!)
- 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 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: Bn -> B where B = {0,1}
  - Find evaluation **x** = (x1,…xn) from Bn such that F(**x**) = 1
- CNF: $f(\mathbf{x}) = c_1(\mathbf{x}) \,\&\, c_2(\mathbf{x}) \,\&\, \ldots \,\&\, c_m(x)$
  - 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 classes, smart selection of local search direction

REPRESENTATIONS

- Straightforward **bit-string** - individual is Boolean vector x
- **Floating-point - e**ncode 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*k for k-SAT with m clauses
  - A special fitness is needed that reflect global inconsistencies in assignments
- **Path-based** – visit clauses and select !1 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

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

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

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

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

FITNESS

- f itself – not that good
- Number of satisfied clauses

$$f_{MAXSAT}(x) = c_1(x) + \ldots + c_m(x),$$

- Weight the problematic **clauses**

$$f_{SAW}(x) = w_1 \cdot c_1(x) + \ldots + w_m \cdot c_m(x)\,.$$

  - Update after some iterations (x* is current fittest individual):

$$w_i \leftarrow w_i + 1 - c_i(x^*),$$

FITNESS CONTD.

- Refining function – identify problematic **variables** via clauses
  - v_j is weight for x_j
    - High positive – x_j tends to be 1
    - High negative – x_j tends to be 0
  - K(y) = -1 for y=0
  - K(y) = 1 for y = 1
  - Updates for v:

$$f_{REF}(x) = c_1(x) + \ldots + c_m(x) + \alpha \cdot r(x)$$

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

$$v_i \leftarrow v_j - K(x_j^*) \cdot |U_j(x^*)|$$

    - x* current fittest individual
    - $U_j(x^*)$ set of unsatisfied clauses for variable j

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

population = 4 population = 10

OTHER TASKS – SCHEDULING (LIKE MATFYZ)

- 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 constrains about the schedule quality
- Hard constrains
  - Must respect in operators, otherwise too many inadmissible solutions are generated
  - Teachers constrains, when, where what to teach, …

OTHER TASKS – JOB SHOP SCHEDULING (LIKE SKODA FACTORY)

- 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 permutation of products order. Decoder must choose plans for parts. Simple representation, can use TSP-inspired crossovers. But show 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

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 Nichael 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

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

TREE-BASED GP

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

INITIALIZATIONS

- 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

MUTATIONS

- 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, …)

CROSSOVERS

- Swap two subtrees
- Uniform crossover – GPUX (Poli, Langdon)
  - At early stages GPUX 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.