EVA II

NAIL 086 - 2014/15

Roman Neruda

Themes, outlines, resources.

INTRODUCTION

Literature

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

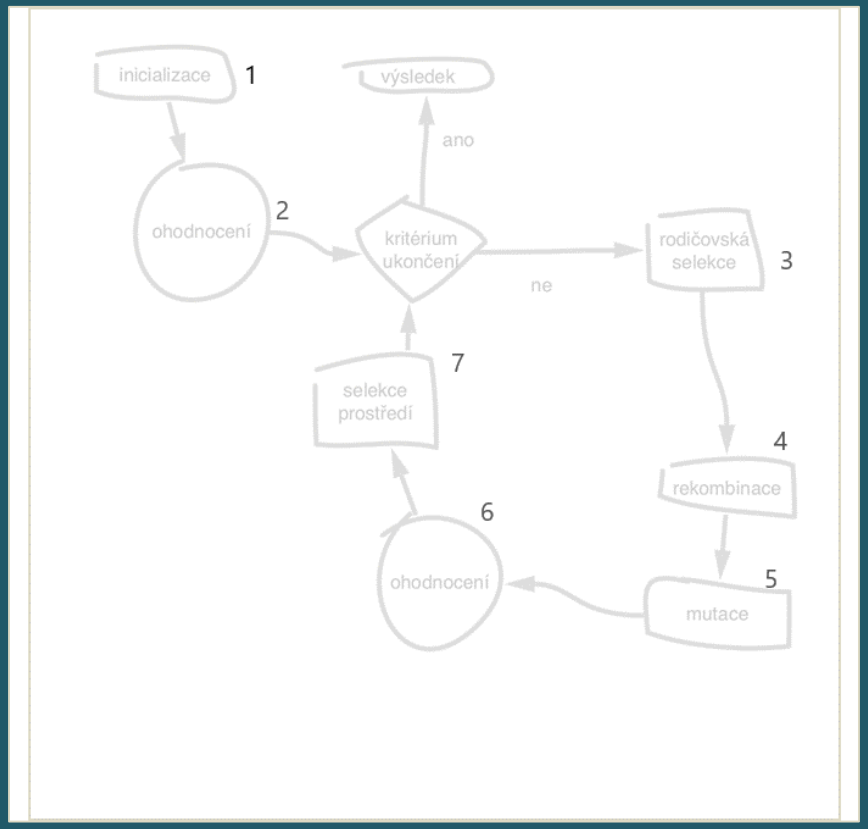
Themes

- Theory of GA revisited – (in)finite populations, Markov chains
- Black box optimization
- Evolutionary programming
- Genetic programming
- Scatter search, Tabu search
- Neuroevolution – grammatical encoding, cellular encoding, NEAT, HyperNEAT
- Artificial life, PSO, ants, and similar algorithms
- Differential evolution
- Memetic algorithms
- Tuning, control, meta-evolution, co-evolution

Everything can be changed

TUNING, CONTROL, META-EVA

General EVA scheme



A. Init
   1. Create initial population P(0) (more or less randomly)
   2. Evaluate

B. In cycle create P(t+1) from P(t):
   3. Parental selection
   4. Recombinations
   5. Mutations
   6. Evaluate new individuals
   7. Environmental selection creates P(t+1) based on P(t) and new individuals

Tuning vs. control

- EA has various parameters (population size, probability of operators, tournament size, elitism rate, ...)
- Common practice is to ***tune*** those parameters for a task at hand
  - Trial and error, grid search
  - Experience and good luck
- They are not independent & it is usually not possible to do exhaustive search really
- One possible solution is to not consider them static but dependent on time or other algorithm properties (e.g. fitness variance in population), and change them – ***control***.

How?

- Fixed strategy
  - Time-dependent change of parameter
  - Deterministic
    - Decrease mutation probability 0.1% each generation
  - Stochastic
    - Simmulated annealing
- Adaptive strategy
  - Based on information about the quality of solution, statistical properties of population, …
- Self-adaptive
  - Evoluion of evolution
  - Mutation variances in genome of ES

Ex 1: Reprezentation

- Adaptive change of representation (encoding of individual) based on population diversity:
  - GA with binary encoded real numbers, unknown precision is needed, so start with small number of bits per real
  - Run GA iteratively, every run is stopped when population diversity goes bellow 1
    - Measured by Hamming distance between the best and the worst individual in the population.
  - Save the best individual as reference, and run GA again
  - If the second run produces the same best individual, increase number of bits per real and start again

Ex 2: Mutation

- Gaussian mutation of real vectors:
  - $x(t+1) = x(t) + N(0,s)$, where s is variance
- Deterministic change:
  - $s(t) = 1 – 0.9\ (t/T)$, where t=0, ... T
- Evolutionary strategies – first kind:
  - Heuristic observation from Berlin – the optimal success rate of mutation is 1/5,
    - If the current rate is bigger, increase s about 10%,
    - If the current rate is smaller, decrease s about 10%
- Evolutionary strategies – second kind (sebe-adaptace):
  - Put s (for every x(t)) into the genom and evolve

Ex 3: Selection

- Boltzman tournament:
  - Probabilty that the worse individual can win will depend exponentially on the ratio of fitness difference and temperature
  - Temperature decreases with time
  - If we stuck in local optima, increase the tempurature
- Simmulated annealing as local search:
  - Like hill climbing, but with boltzman probability of acceptance of worse solution

Ex 4: Population

- Indirect control of population size depending on the fitness of individuals
  - When new individual is generated, we define its life span proportional to its fitness
  - After the defined number of generations, the individual dies
  - Better individuals live longer, thus the system supports spread of successful individuals
  - Population size is thus only a derived, dynamic, and not so important thing

Beyond schemata

THEORY REVISITED

Exact models of simple GA

- 90s: Vose, Lepins, Nix, Whitley, ...
- Attempt to mathematically study:
  - How exactly the population looks like
  - Describe mapping of transition from one population to the next
  - Properies of this mapping
  - Asymptotic behavior of simple GA
- Infinite populations
- Finite populations

Let us simplify the (already simple) plain vanilla GA - SGA

- Random initial population of length l binary strings x
  - Compute Fitness f(x)
  - Repeat until new population is filled:
    - Select 2 individuals,
    - Cross them over with probability p_c
    - Discard one of them (!) <- this is the simplification
    - Mutate each bit with probability p_M
    - Insert new (one) individual to new population
- Repeat until a good enough individual x is found

Formalize the SGA

- Each binary string is represented by a number from $0..2^l$
  - 00000111 is 7
- Population in time t is represented:
  - By two vectors: p(t) a s(t) of lenght 2^l
  - p_i (t) represents the part of population t occupied by a string (relative frequency)
  - s_i (t) is probability of selection of string i
- p(t) defines the population composition
- s(t) defines selection probabilities

The capital G operator

- We have fitness f, define matrix F(i,j):
  - F(i,i) = f(i); F(i,j) = 0 pro i<>j
- Then $s(t) = F\ p(t)/(\Sigma\ F(j,j)\ p_j(t))$
  - (that's in fact a definition of fitness-proportional selection)
- Our dream is to define operator G that would realize the SGA iteration:
  - i.e. p(t+1) = G p(t), or similarly s(t+1) = G s(t)
- If we have G, we iterate it from initial population, and we know everything about SGA (!)
- Let us decompose G = F°M (F fitness, M mutation and crossover)

First let G=F

- E(x) expected value
- E(p(t+1)) = s(t)
- s(t+1) ~ F p(t+1) (just multiplicative constant c)
- Thus: E(s(t+1)) ~ F s(t)
  - In case of (small) finite population, the statistical errors can cause deviations from expected values E(.)
  - The bigger population, the smaller deviation
  - Inifite populations are exact, albeit a bit impractical

Now let G=M

- M is a recompination operator (including both crossover and mutation)
- The derivation requires some probability skills and lots of time:
  - Let us use a handy function r(i,j,k)
  - r(i,j,k) ... Probability that offspring k is created from parents i and j
  - When we have r(i,j,k), we can compute p(t+1)
  - Well, one part of vector after another, and also in expected values, but that's why we now consider infinite populations:
- $E(p_k(t+1)) = \Sigma_i \Sigma_j \, s_i(t) \, s_j(t) \, r(i,j,k)$

R(i,j,0)

$$r_{i,j}(0) = \frac{(1-p_m)^l}{2}\left[\eta^{|i|}\left(1-p_c+\frac{p_c}{l-1}\sum_{c=1}^{l-1}\eta^{-\Delta_{i,j,c}}\right)\right.$$

$$\left.+\eta^{|j|}\left(1-p_c+\frac{p_c}{l-1}\sum_{c=1}^{l-1}\eta^{\Delta_{i,j,c}}\right)\right].$$

A zbytek analogicky

- Derivation of r(i,j,k) is possible, this is r(i,j,0)

Results

- SGA works by applying G to each population in time, it is a ***dynamical system***, and vectors p(t) or s(t) are points of the trajectory of such a system.
- What are the fixed pointsJ? (well, that exactly we still do NOT know)
  - Fixed points of F are poplations with identical fitness
  - Stable fixed point of F: maximal identical fitness
  - (The only) fixed point of M: identical probabilities s (or, the same relative frequencies of individuals p)
  - Combination of shaking by M and focusing by F - *punctuated equilibria (known from biology)*

Finite populations

- Markov chains:
  - Stochastic process in discrete time
  - System has states, the probability of going from one state to another depends only on the first state (no memory of previous states)
  - Table of transition probabilities between all states represents a complete description of such a system
- Wow, let us define SGA over fitine populations as Markov chain

States

- Each state of our Markov chain is one particular populations.
- There is quite a lot of populations
- N = number of populations containing n individuals of length l:
  - $N = (n+2^l-1)$ over (2^l-1)
- Matrix Z
  - Columns are populations
  - Z(y,i) = number of individuals y in population i
- (thus, columns of Z are states of our Markov chain)

Transition matrix

- Q is a matrix of transition between populations, it has NxN elements
- So, e.g. for n=l=8 , Q has 10^{29} numbers!
- Q can be derived, but seldom used

$$Q_{i,j} = n! \prod_{y=0}^{2^l-1} \frac{\left[\mathcal{M}\left(F\vec{\phi}_i/|F\vec{\phi}_i|\right)_y\right]^{Z_{y,j}}}{Z_{y,j}!}.$$

What have we achieved, really?

- Exact analysis of SGA, given f
- In practice intractable, though
- Asymptotical results – provide insight into convergence and SGA behavior, although the composition of F and M into G is still open.
- **A correspondence exists between infinite pop. SGA and finite pop. SGA (limit approach)**

Genotype is phenotype, crossover is evil

EVOLUTIONARY PROGRAMMING

Evolutionary programming

- L. Fogel, 60s (older than Holland GAs)
- The ultimate goal was to evolve „artificial intelligence“:
- Agent is situated in an environment
- Agent should predict environment state
- Agent should derive a suitable action w.r.t the environment state
- Agent was repsented by a finite automaton
- Environment was: sequence of finite alphabet symbols

Finite automaton

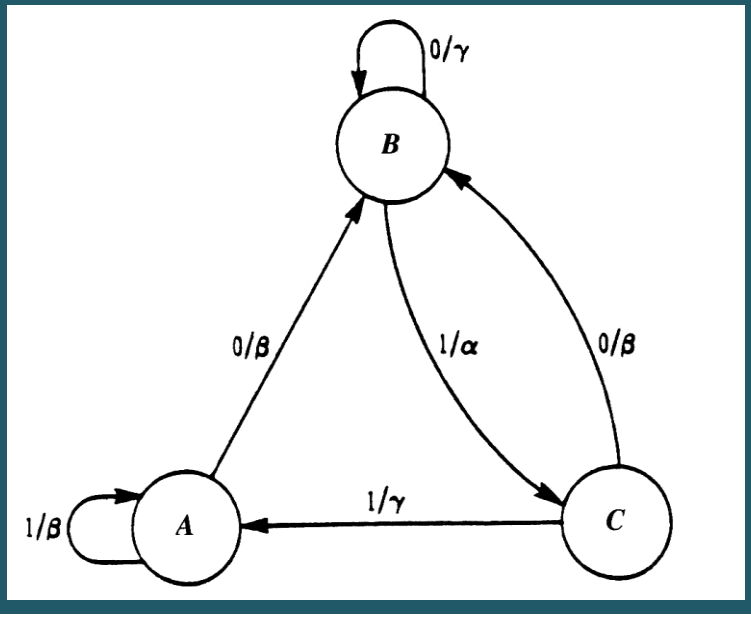


Table 3-1 The Response of the Finite-State Machine Shown in Figure 3-2 to a String of Symbols. In This Example, the Machine Starts in State C.

Present State	C	B	C	A	A	B
Input Symbol	0	1	1	1	0	1
Next State	B	C	A	A	B	C
Output Symbol	β	α	γ	β	β	α

EP over FA

- Population of finite automata – (remember, genotype = phenotype)
- Inputs are presented to automaton
- Outputs are compared to next symbol in the sequence
- Fitness: success of the prediction (measured differently)
  - Absolute error
  - Mean square error

EP over FA cont.

- Creating new automata: ***mutation***
  - Change output symbol
  - Change successor state
  - Add a state
  - Remove state
  - Change initial state
- ***No crossover*** (!)
- Half of population replaced by new automata
- Fitness often considers the size of automaton

Famous Ex.: Primes prediction

- The goal is to predict a sequence of 0s and 1s
  - Where 0 means that number corresponding to this position is not a prime.
  - i.e. learning the sieve of Erathosthenes
  - 11101010001010001010001000010 …
- Iteratively:
  - Sequence of fixed length is learned,
  - Then, increase the size by one symbol …
- Fitness:
  - Point for each guessed symbol in sequence
  - Negative fitness part : - 0.01 * N (=number of states)
- Automata were evolving simpler and simpler, ultimately being 1-state and generating 0 all the time

Primes cont

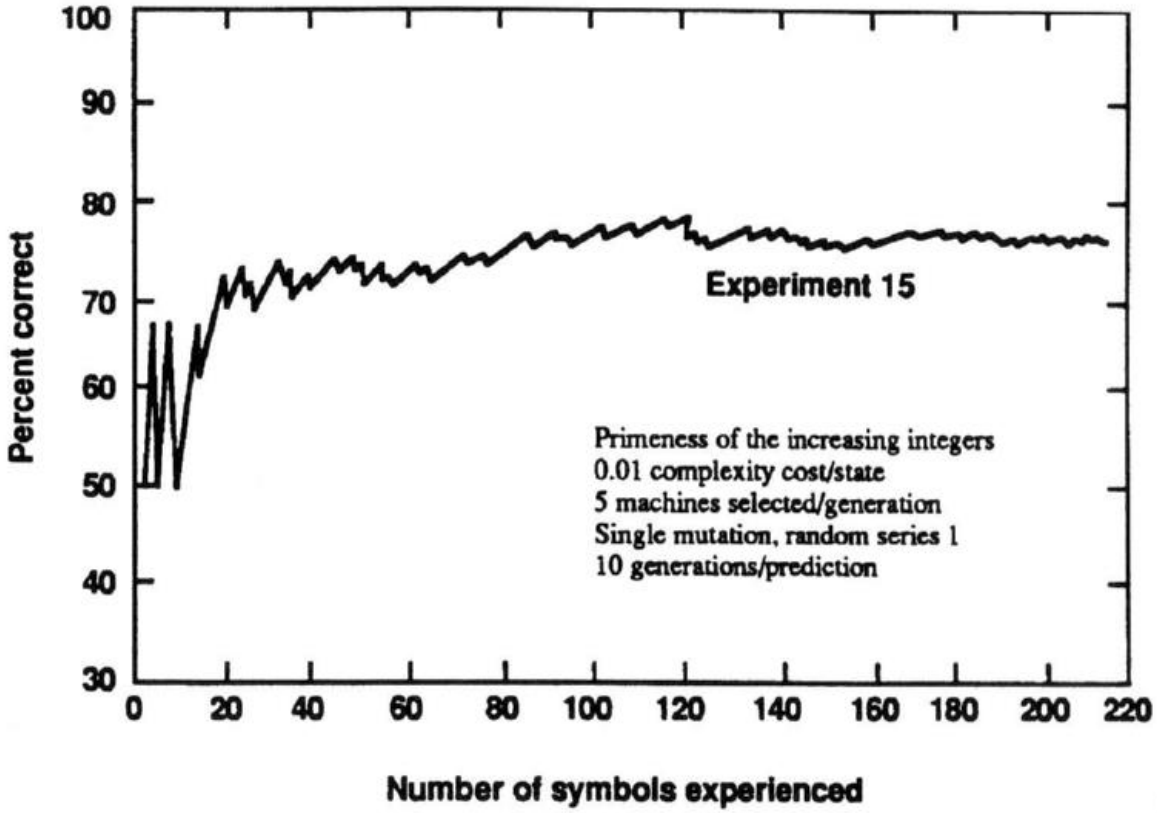


Figure 3-3. The cumulative percentage of correct predictions in the first 200 prime numbers (from Fogel et al., 1966, p. 37).

EP today

- Benevolent to encoding of individuals
  - Encoding should be natural for the problem at hand
  - Often Genotype = Phenotype
- A collection of "smart"mutations, tailored to problem and encoding
- No crossover
- Parental selection: every is selected once
- Environmental selection: from parents and offsprings, new population is selected by tournament

EP over reals

- Encoding
  - Vector of reals
  - Contains variances of mutations
- Mutation
  - Small random change according to normal distribution with corresponding variance
- Meta-evolution
  - Parameters that are used in algorithm, are modified by the algorithm
- Selection: all are parents, then tournament

EP over reals

procedure Meta-EP
t←0
Inicializuj P_t N náhodně vygenerovanými reálnými vektory $\vec{x}_t = (x_{1t},\ldots,x_{nt},\sigma_{1t},\ldots,\sigma_{nt})$
Ohodnoť jedince v populaci P_t
while (neplatí kritérium ukončení) do
    for i ← 1, . . . , N do
        k rodiči $\vec{x}_t$ vygenruj potomka $\vec{y}_t$ mutací:
        for j ← 1,...,n do
            $\sigma_j' \leftarrow \sigma_j \cdot (1+\alpha \cdot N(0,1))$ $x_j' \leftarrow x_j + \sigma_j' \cdot N(0,1)$
        end for
        Vlož $\vec{y}_t$ do kandidátské populace potomků P'_t
    end for
    Turnajovou selekcí vyber P_{t+1} z rodičů P_t a potomků P_t'
    Zahoď P_t a P_t'
    t←t+1
end while
end procedure

EP vs GA

- Crossover:
  - Exchange of building blocks – ideal case
  - Strange big random mutation – not respecting the encoding
- Jones 1995: headless chicken experiment:
  - The success of crossover was tested by comparing the traditional crossover with ***random*** crossover (crossing the individual over with random string)
  - There is no crossover without the idea of crossover, do not call headless chicken a chicken, although it has many chicken features.

Headless chicken

- In cases with clear idea of building blocks, the crossover is better than random crossover
- In cases, where the random crossover is better, it is in fact a macro-mutation
- In such a case there are no clear building blocks
  - Wrong encoding?
  - Wrong crossover for this encoding?
  - Difficult problem for crossover?
- The moral: Is our chicken headless?

GA vs EP: experiment

Table 4.1 The Number of Parameters, The Binary Coding Length, and The Functions Studied in Schraudolph and Belew (1992). These Follow Previous Efforts By De Jong (1975). The Operation $[x_i]$ in $f3$ Returns The Greatest Integer Less Than or Equal To x_i. The $N(0, 1)$ in $f4$ Represents A Standard Gaussian Random Variable.

Function	Dimension	Bit Length
$f1$	3	10
$f2$	2	12
$f3$	5	10
$f4$	30	8
$f5$	2	17

Function	Parameter Range
$f1: F(x) = \sum_{i=1}^{3} x_i^2$	$[-5.12, 5.12]$
$f2: F(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.048]$
$f3: F(x) = \sum_{i=1}^{5} [x_i]$,	$[-5.12, 5.12]$
$f4: F(x) = \sum_{i=1}^{30} i x_i^4 + N(0, 1)$	$[-1.28, 1.28]$
$f5: F(x)^{-1} = 1/K + \sum_{j=1}^{25} f_j(x)^{-1}$.	$[-65.536, 65.536]$

$$f_j(x) = c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6.$$

where $K = 500$, $c_j = j$, and

$$[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \dots & 32 & 32 & 32 \end{bmatrix}$$

$f6$: $F(x, y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$,

$f7$: $F(x, y) = x^2 + 2y^2 - 0.3(\cos(3\pi x)\cos(4\pi y)) + 0.3$,

$f8$: $F(x, y) = x^2 + 2y^2 - 0.3(\cos(3\pi x) + \cos(4\pi y)) + 0.3$.

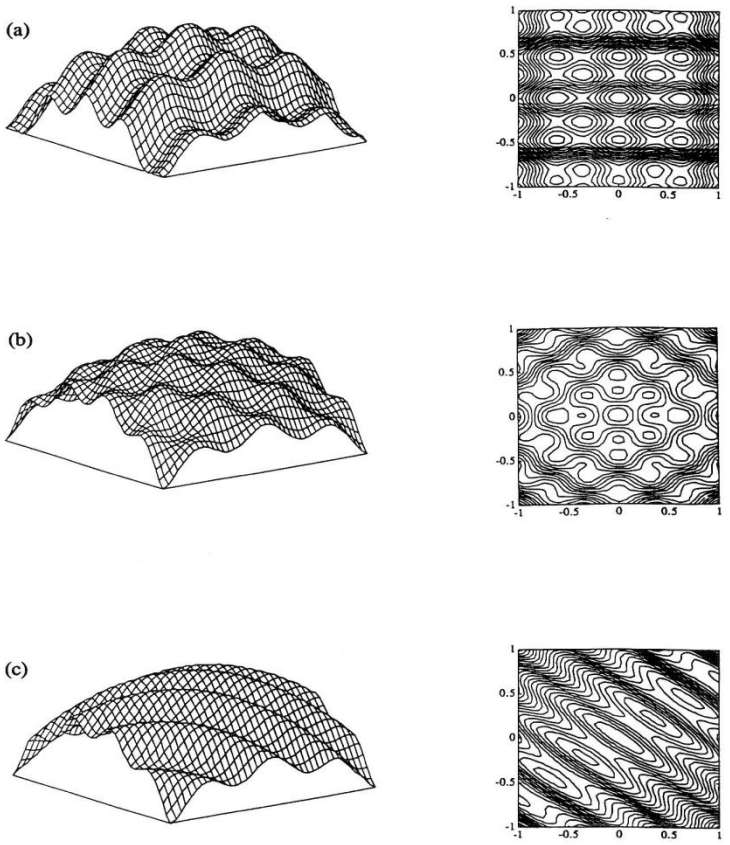


Figure 4-17 Inverted and contour plots of the three Bohachevsky functions studied in Fogel and Stayton (1994). (a) $F(x, y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$. (b) $F(x, y) = x^2 + 2y^2 - 0.3(\cos(3\pi x)\cos(4\pi y)) + 0.3$. (c) $F(x, y) = x^2 + 2y^2 - 0.3(\cos(3\pi x) + \cos(4\pi y)) + 0.3$.

GA vs EP exp pokr

Table 4-3 Results for the Best Score in the Population and the Mean of All Parents' Scores After 10,080 Function Evaluations, Averaged Over 10 Trials With the Genetic Algorithm Techniques (Both With And Without Dynamic Parameter Encoding) and 500 Trials With "Evolutionary Programming" (after Fogel and Stayton, 1994). Evolutionary Programming Outperforms Both Genetic Methods On Functions $f1, f2$, and $f6$–$f8$, and Yields Comparable Performance on $f3$–$f5$. The Values in Parentheses Indicate the Standard Deviations

		Average Best	Average Mean
$f1$:	EP	3.149×10^{-66} (2.344×10^{-129})	1.087×10^{-65} (1.794×10^{-128})
	DPE	1.056×10^{-11} (1.072×10^{-21})	3.618×10^{-10} (1.060×10^{-18})
	GA	2.836×10^{-4} (4.587×10^{-8})	6.135×10^{-1} (1.627×10^{-1})
$f2$:	EP	1.215×10^{-14} (3.357×10^{-26})	8.880×10^{-14} (2.399×10^{-24})
	DPE	2.035×10^{-2} (3.315×10^{-3})	8.785×10^{-2} (8.090×10^{-3})
	GA	2.914×10^{-2} (6.280×10^{-4})	1.722×10^{0} (2.576×10^{0})
$f3$:	EP	0.0 (0.0)	0.0 (0.0)
	DPE	0.0 (0.0)	0.0 (0.0)
	GA	0.0 (0.0)	1.307×10^{0} (1.172×10^{-1})
$f4$:	EP	-2.575×10^{0} (7.880×10^{-1})	-4.274×10^{-1} (3.206×10^{-2})
	DPE	-2.980×10^{0} (1.009×10^{-1})	4.704×10^{-1} (1.283×10^{-1})
	GA	-4.599×10^{-1} (4.265×10^{-1})	1.312×10^{1} (2.941×10^{0})
$f5$:	EP	4.168×10^{0} (9.928×10^{0})	4.194×10^{0} (1.022×10^{1})
	DPE	3.502×10^{09} (1.265×10^{1})	1.642×10^{1} (7.101×10^{2})
	GA	9.980×10^{-1} (3.553×10^{-15})	1.021×10^{1} (7.165×10^{1})
$f6$:	EP	5.193×10^{-96} $(1.348 \times 1010^{-188})$	9.392×10^{-94} (4.410×10^{-184})
	DPE	1.479×10^{-9} (1.460×10^{-18})	8.340×10^{-7} $(4.649 \times 1010^{-14})$
	GA	2.629×10^{-3} (1.103×10^{-5})	4.022×10^{1} (6.467×10^{3})
$f7$:	EP	8.332×10^{-101} (3.449×10^{-198})	2.495×10^{-99} (3.095×10^{-195})
	DPE	2.084×10^{-9} (6.831×10^{-18})	6.520×10^{-7} $(7.868 \times 1010^{-14})$
	GA	4.781×10^{-3} (2.146×1010^{-5})	3.541×10^{1} (2.922×10^{3})
$f8$:	EP	1.366×10^{-105} (4.479×10^{-208})	3.031×10^{-103} $(2.122 \times 1010^{-203})$
	DPE	1.215×10^{-5} (5.176×10^{-10})	3.764×10^{-1} (9.145×1010^{-1})
	GA	2.444×10^{-3} (4.511×10^{-5})	2.788×1010^{-1} (1.368×10^{-3})

Skip the middle man

GENETIC PROGRAMMING

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)

Evolution of programs

- 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
- ***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

Variations of GP

- ***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, …)
- ***Crossover:***
  - Uniform crossover on a sub-tree

Variations of GP

- ***Initialization:***
  - 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

ADF

- Automatically defined functions
  - Subprograms – necessary to achive modularity, higher complexity, represent symmetries in the problem
- Characterized by their arity,
- Have limited (or different) terminal and non-terminal sets
- A program typically has main() routine and one or more ADF subtrees
  - ADF call becomes a new non-terminal in the main program
- GP operators work on mains and ADFs separately, the routines are not mixed together
- Specialized code constructs have been proposed based on ADF approach: automatically defined loops, iterations, recursions …
- Alternative approach – co-evolution of population of mains and ADFs

Bloat of GP programs

- Programs in GP have tendencies to grow in size - ***bloat***
- In nature, the extensive growth hits some physical limit that prevents excesses
- In GP we must fight bloat explicitly:
  - Limit tree size (no. of nodes), limit tree depth:
    - Penalize it as a negative term in fitness
    - Or, check and repair/eliminate new individuals
  - Anti-bloat operators:
    - Watch mutation and crossover so that do not enlarge much
    - Or, special mutations making trees smaller

Way dowm: linear GP

- 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

- Graph-based 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 (EP made it illegal)

A rollercoaster of hopes and obstacles

NEUROEVOLUTION

Learn NN by EA

- First attempts in late 1980s
  - Learn parameters (weights)
  - Learn architecture (topology, connections)
  - Learn both architecture and weights at once
  - Learn other things (activation function, …)
- Panacea for reinforcement learning – where it is not possible to use supervised learning (robotics)
- Hybrid methods – combine EA with local search, back propagation, …

Learn weights by EA

- Straightforward
  - Encode weights into fixed-length vector
  - Use floating point GA, evolutionary strategies, …
  - Standard operators, well known approaches
- Still, will be slower than specialized gradient based local search algorithms, such as back propagation
- + can be parallelized
- + can be used for reinforcement tasks
- Recent renewal of interest for evolution of weights
  - Actually works quite comparatively
  - use mini batches training strategy for fitness

Learn the architecture

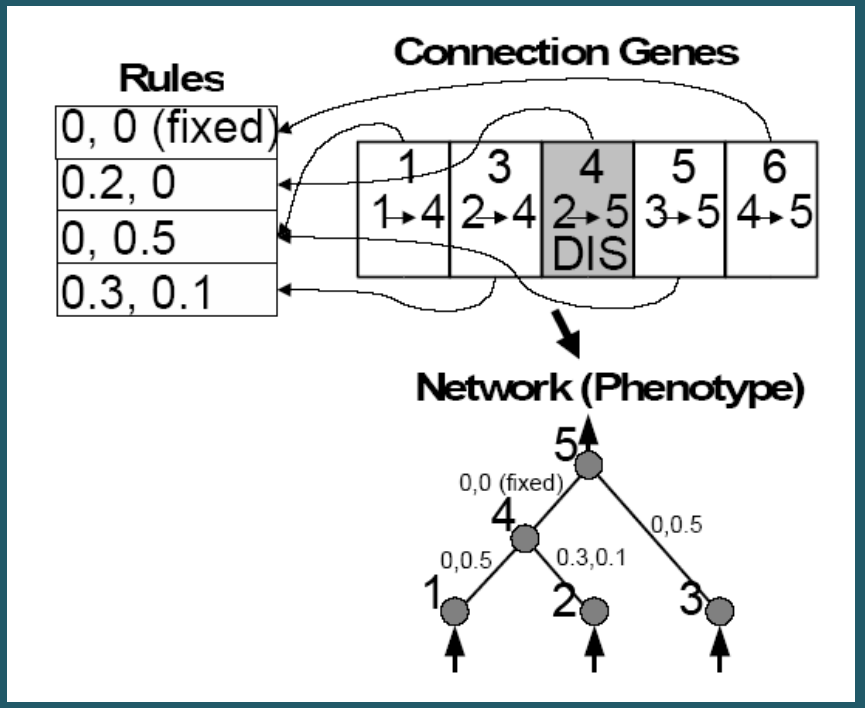
- Fitness = performance estimation of the network
  - build the network, initialize, try to learn (by BP, or another EA, maybe), better to do it more times,
- Direct encoding
  - The structure of connections is represented as binary matrix or similar natural representation
  - Then the individual is either linearized matrix – i.e. long binary vector (standard operators), or the matrix itself (special 2D operators)
- Grammatical encoding
  - Kitano, 1990 proposed to represent binary matrices by means of simple formal 2D grammars that will induce the matrix. EA then evolves the grammar rules
  - compact, logarithmic, but too indirect

Grow the architecture

- Simmulated growth of network connections in 2D planes
  - Early attempts in evolutionary robotics
  - Not scalable
- Celular encoding
  - Gruau proposed to use GP to represent architecture
  - GP is a program how to grow a network by means of operations:
    - Add neuron, split neuron in a serial way, split neuron in a parallel way, swap synapse, …

NEAT

- K. Stanley, 2002 – Neuroevolution of augmenting topologies
- NN is represented as a list of edges, each edge:
  - Information about its vertices,
  - Weights, disabled flag, and
  - Globally maintained ***Edge ID***.



NEAT cont.

- Crossover edges with the same ID only, the rest remains unchanged
  - This crosses over only edges with the same evolutionary origin, does not mess the overall structure, no headless chicken here
- Define similarity measure on vectors of edge IDs (how much two network architectures differ)
  - Niching – similar networks are considered to be the same species. Fitness is computed as relative in each species.
  - This solves the problem that structural changes disrupt the fitness, but are necessary for exploration.
  - Niching allows to protect newly created topologies before the weights are tuned.
- Later the same principles were applied in HyperNEAT.

Urban legends and fake news

MEMETIC ALGORITHMS

R. Dawkins of Selfish gene

- Meme
  - (neo)-Darwinist look at cultural and social areas
  - Evolution, spread and survical of various ideas, thoughts, concepts, … in human society
  - Something like biological evolution, but without DNA
- Two types of memetic algorithms
  - Simulation of memes in society
    - Bordering with psychology, literature (urban legends), religionism, cultural anthropology
  - Utilizing the concept of memetic (sometimes called cultural) space for meta-heuristics in EA
    - Let us take a closer look on this memetic algorithm

Memetic algorithm

```
t = 0;
initialize(P(t=0));
P(t=0).localSearch();
evaluate(P(t=0));
while isNotTerminated() do
    P(t) = selectIndividuals();
    mutate(P(t));
    P(t).localSearch();
    evaluate(P(t));
    P(t+1) = buildNextGenerationFrom(P(t));
    t = t +1;
end
```

Memetic algorithm

- In fact, it is a local search nested inside classical EA cycle
- Local search:
  - Hill climbing
  - Simmulated annealing
  - Gradient search (such as back propagation for learning neural networks)
  - …

How to handle a result

- Lamarckism
  - When local search finds a better individual, let us take it
  - This is not kosher from darwinistic point of view, we have changed genotype based on phenotype changes
- Baldwinism
  - When local search finds a better individual, let us take the fitness of the better one, and assign it to the orifinal – unchanged – individual
  - So, no genotype change, Darwin-wise correct, yet the fitness is better, can be interpreted as a potential of the original individual

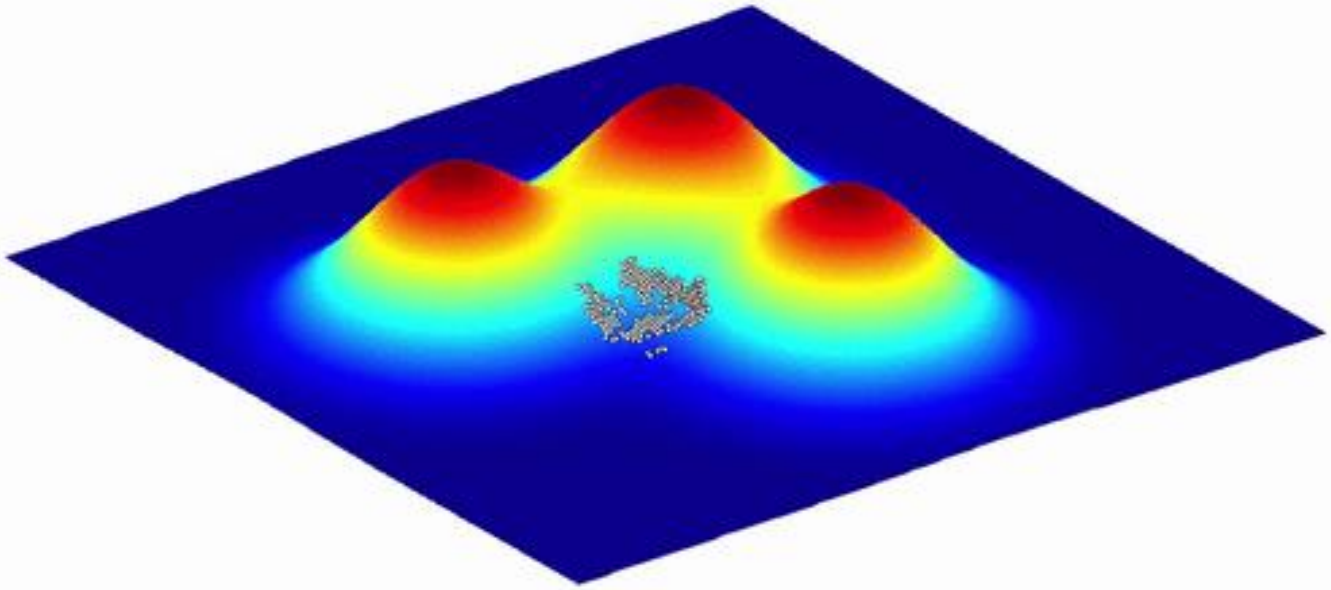
When the world is changing

DYNAMIC FITNESS LANDSCAPES

Fitness landscape

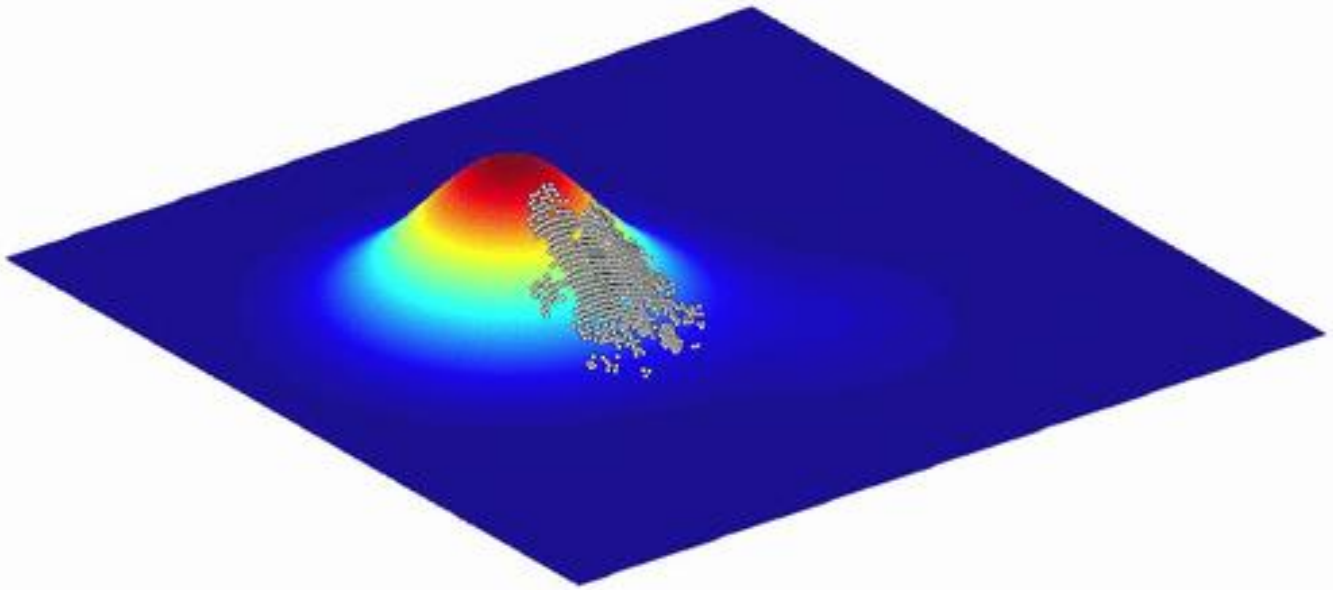
- First introduced by Sewall Wright, 1932
- Graphical representation of fitness dependent on:
  - Genotype
  - Allele frequency
  - (some feature of) phenotype
- They are popular in EA
  - Theretical tool to explore EAs
  - Clear illustration of relation genotype-fitness
    - But higher-dimensions are not intiuitive

Static fitness landscape



Population size, N = 2,304
Mutation rate, μ = 0.05 per trait

Dynamic fitness landscape



Population size, N = 2,304
Mutation rate, μ = 0.5 per trait

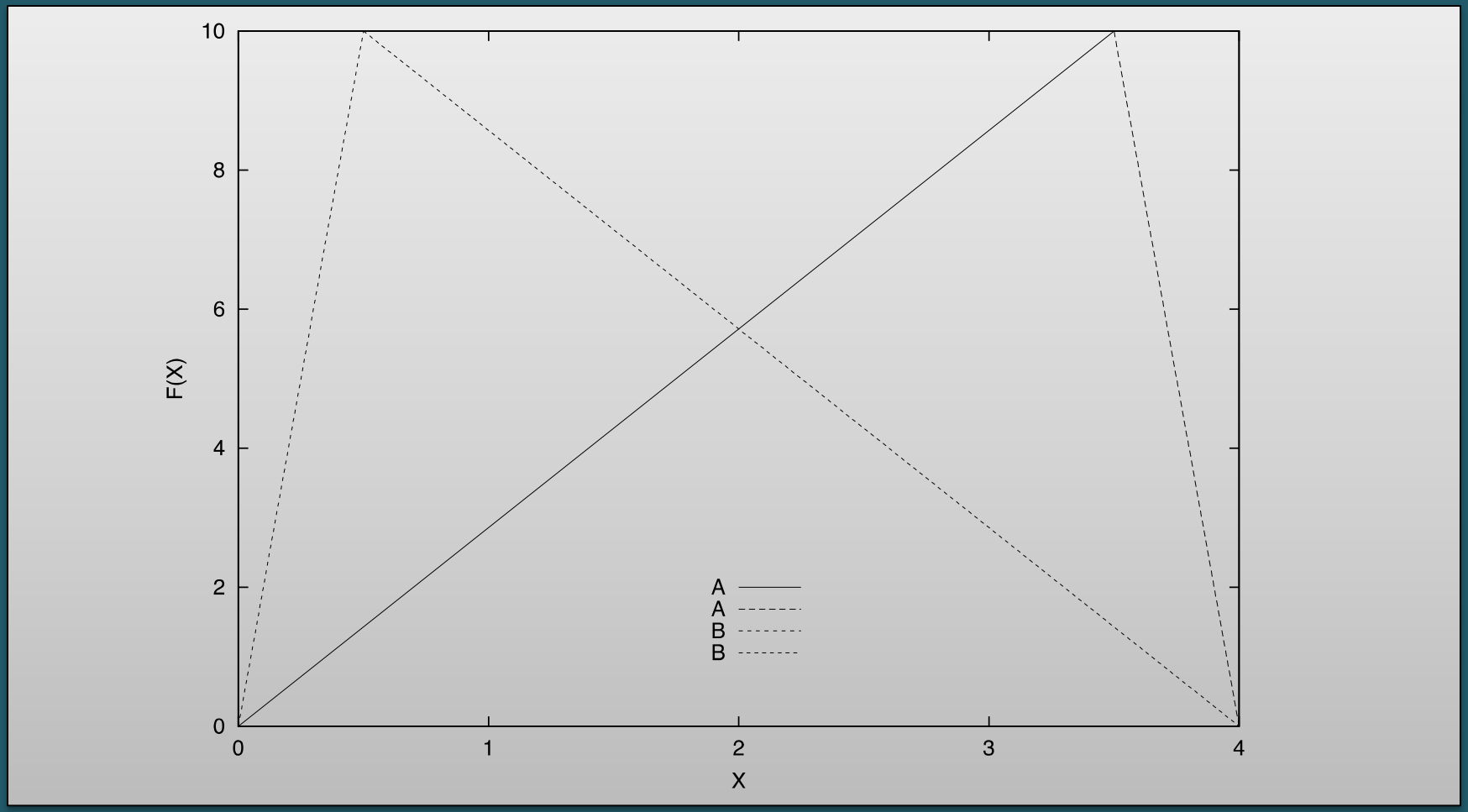
Changing fitness

- Why? – actually quite common
  - Robot in the room, somebody turns on the light
  - It starts raining, but Google car was tested only in California
  - A crisis on stock exchange
  - Turnaments of agents playing some game
- Solution:
  - Human changes something, restart, … very offline
- Can we continue our EA when fitness changes?

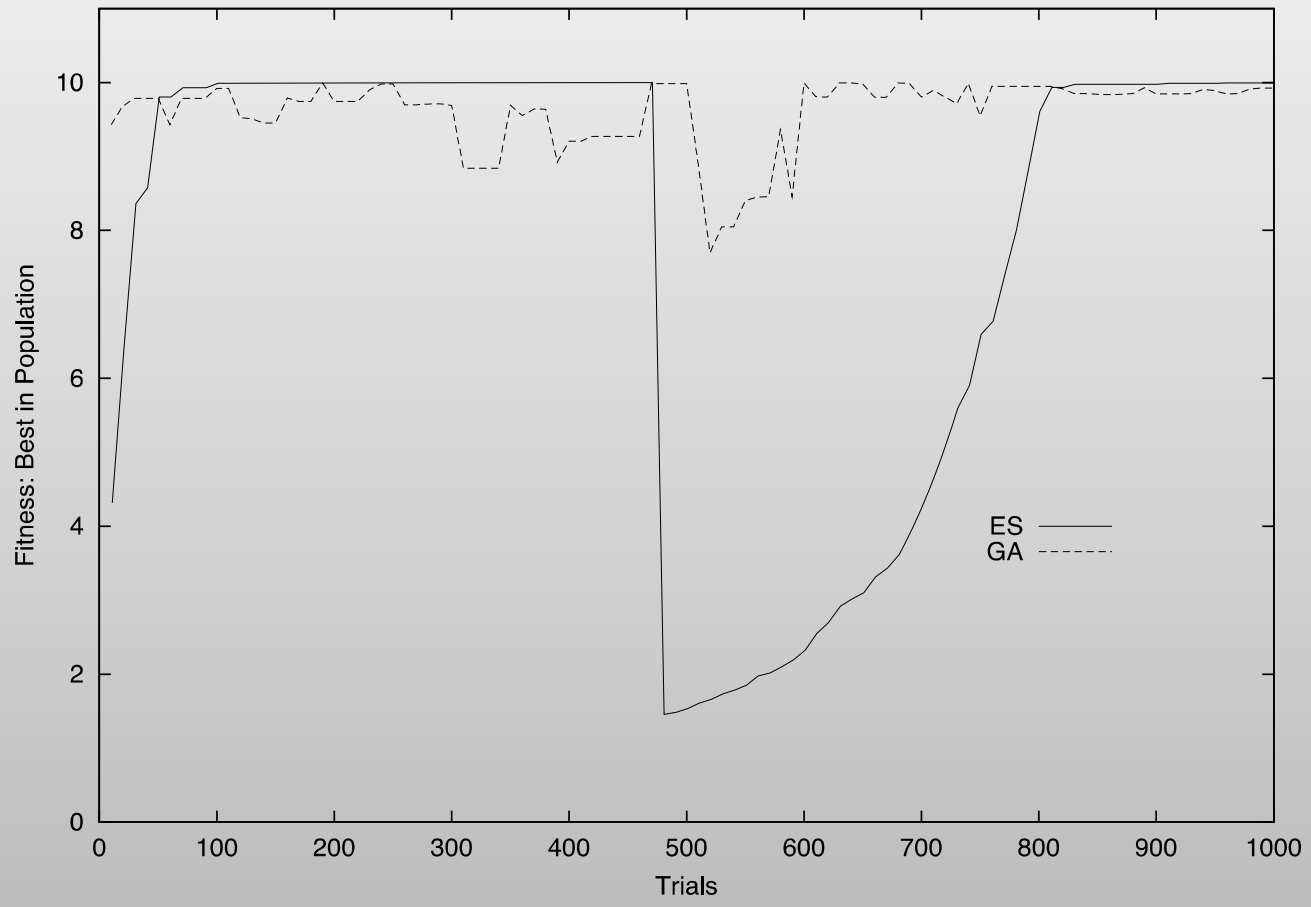
Classical GA with dynamic fitness

- Usually nothing special, it redefines the task
- Typically we want to find an optimal solution of a static task:
  - First explore, but then converge to local extreme which is good enough
  - During the GA run we are losing diversity
- With dynamic fitness we want generality and possibility to change
- Kenneth de Jong:
  - Comparison of (1+10) ES a GA with 10 individuals

Simple fitness landscape with 2 alternating states



Reaction to fitness change



Modification of EA for dynamic f.l.

- Goldberg, Smith
  - Diploid representation works as a long term memory when fitness oscilates
    - Nevertheless, most of our EAs is haploid
- Cobb, Grefenstette
  - Hypermutation
    - When average fitness in population goes down (probably fitness has changed), increase mutation rate quite a lot.
  - Random migration
    - When fitness goes down, generate a relevant amount of new random individuals and put them to population
- Comma ES, makes them more prone to landscape changes

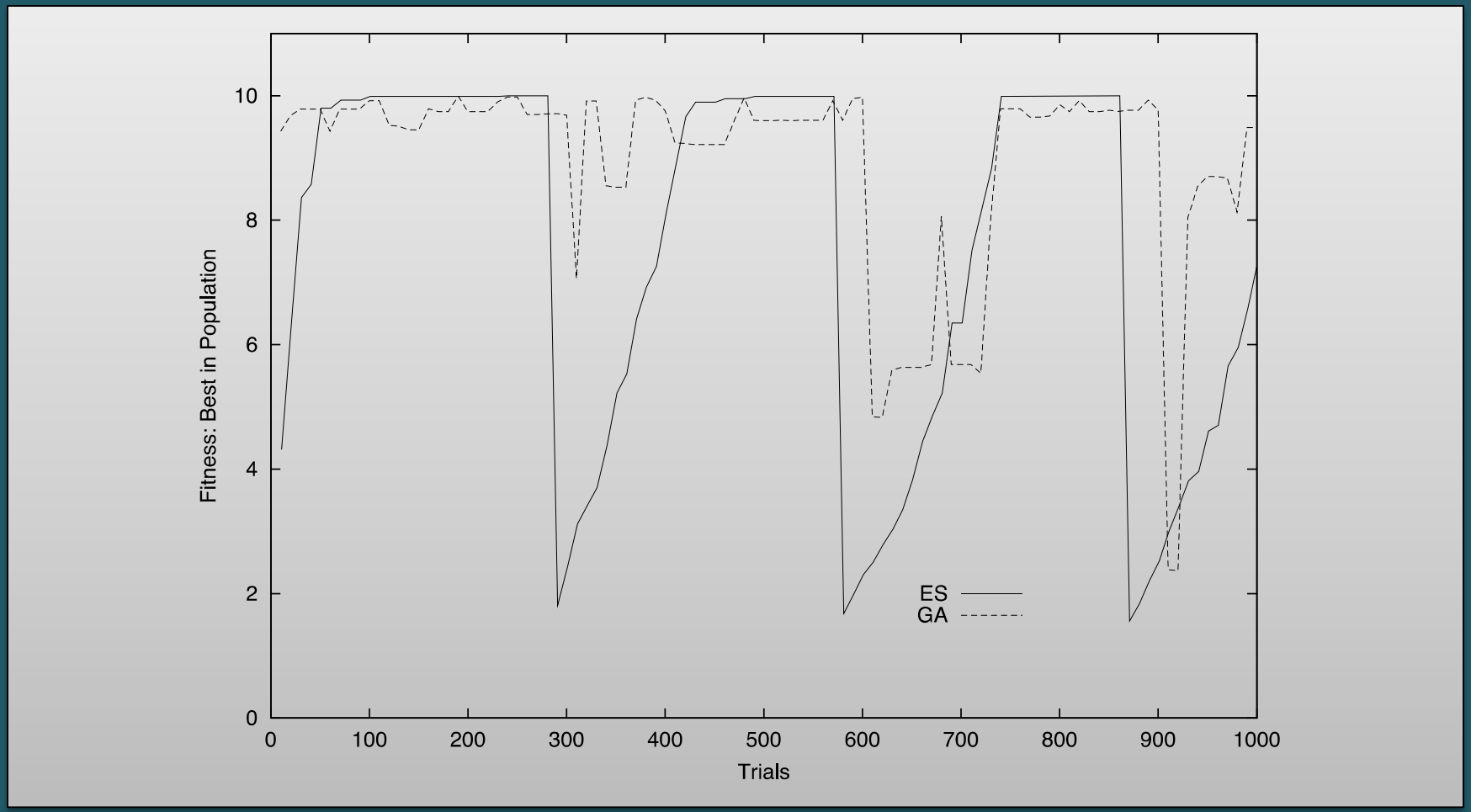
More modifications

- Keep population diverse
  - Decrease selection pressure
  - Crowding, niching
    - Protects population to be overthrown by few fit (at the moment) individuals
  - Sub-populations
    - The ***island model*** – very good just for distributed EA
  - Dynamic species –
    - Utilizing tag-bits for differentiating species
    - Mating only within species

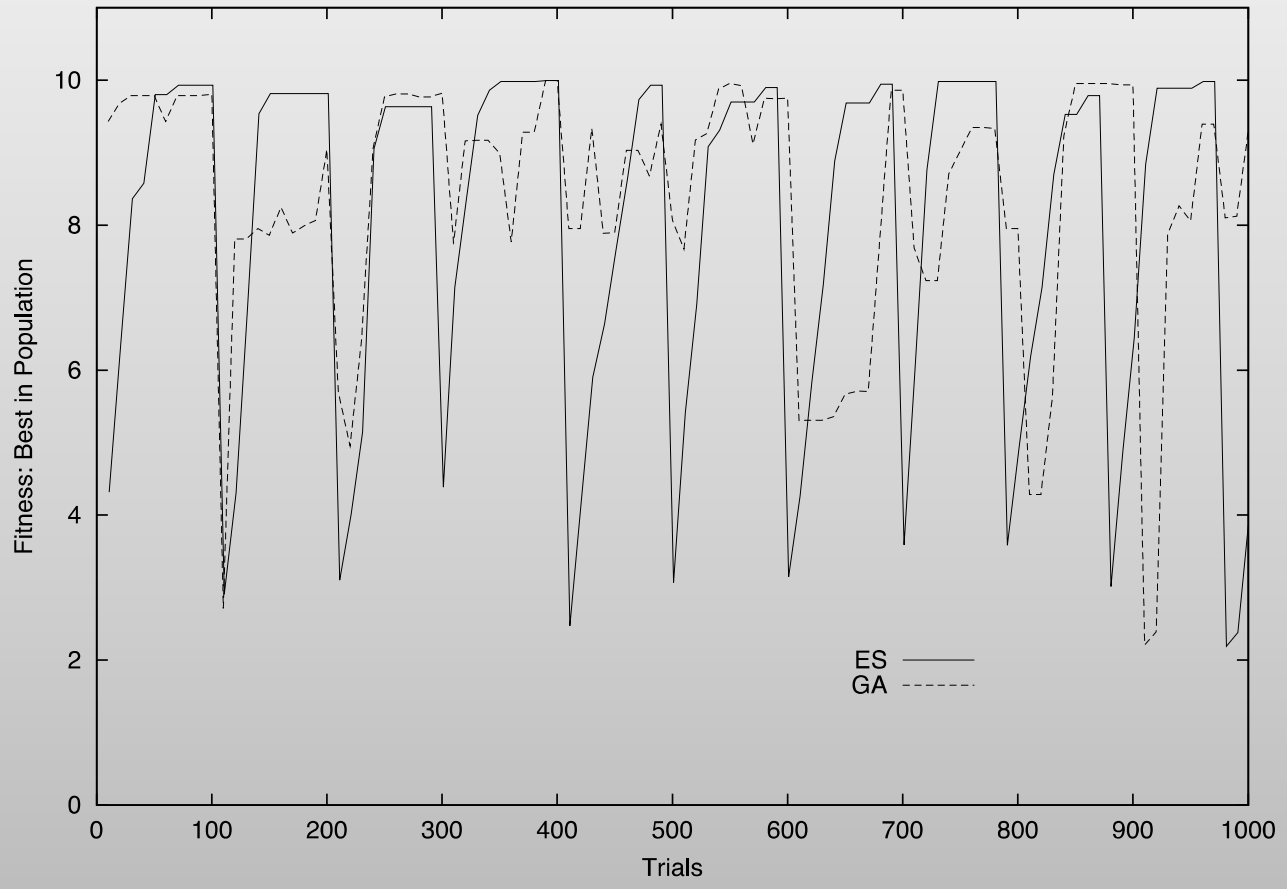
How dynamic landscape are

- When fitness changes rapidly, no solution might be ideal, remember the No-free-lunch theorem
- Small changes are reasonable
  - Robots hardware worns out
  - Slight changes of chemicals in a factory …
- Relevant morphological changes
  - The hills of fitness emerge or disappear, “emerging markets”
- Cyclic changes
  - Times of the year, electricity consumption, …
- Non-continuous catastrophic changes
  - Powerplant explosion, traffic accident, war, …

Oscilating changes



Fast oscilating changes



Nature is complicated, engineers don’t care

BIOLOGICALLY ACCURATE EA

Species

- Even small selection pressure leads to vanishing diversity of population
  - C.f. also Flegr – Frozen evolution concept
- To introduce species or some other mechanism of ***non-random mating*** might save the diversity
- Goals:
  - Better reaction to changes in fitness (old Darwin)
  - Parallel exploration of search space

Niching

- Goals:
  - Slow the convergence towards one optimum
  - Create parallel sub-populations, exploring and converging in different areas
- Recombination is limited to similar genomes only
- Fitness is considered relative for the niche – group of similar individuals
- Crowding: tournaments of similar individuals

Pros and cons

- It works, the declared goals are improved
- It is sensitive to parameter setting:
- How to compute similarity of individuals?
  - Hamming for binary
  - Cf. NEAT and edge IDs
- What is one niche?
  - Dynamic situation
  - Requires a threshold on similarity

Non-random mating

- General mechanism where mating is allowed for individuals that are “close”
- It is necessary to define topology for the individuals
  - Artificial life simulations like this
  - Usually individuals are on a 2D (3D) mesh
  - Island model of GA – usually sub-populations on different machines
- Explicit species and only intra-species mating

Co-evolution

- Evolution of strategies for games – from Tic-tac-toe to checkers …
- Collective type of task
  - Cooperation in team
  - predator-prey
- Context fitness
  - Depends on other individuals in population
  - Or on other species
  - Relationships between co-population can have complex dyamics – deterministic chaos
- ADF in GP
- Sorting networks and evil number sequences – classical

More bio-ideas

- Morphogenetics and generative representations
  - Neuroevolution
  - Celullar automata
  - l-systems
- Agen-based approaches
  - Holland a ECHO
  - Artificial life
- Lamarck – mentioned many times

Mixed bag of – what is also interesting

SEVERAL SEARCH ALGORITHMS

Taboo search

- Approach to local search which tries to not go where it already was
- Algorithm (hill-climbing plus taboo):
  - Current solution is x(t),
  - Generate at random x(t+1) from neighbourhood N(x(t))
  - If f(x(t))<=f(x(t+1)), continue with x(t+1),
  - Else continue with x(t)
  - N(x) is neighbourhood of x,
  - Moreover, I keep a buffer of several last x(t) that will be excluded from N(x)
  - I call this a **Taboo list**
- Taboo set cen be customized to task at hand, may represent more complex constrains

Tabu set

- Short-time memory
  - List of recent previously visited solutions
  - Cannot revisit them while they are in the time span of the list
- Mid-term memory
  - Rules guiding the search towards areas of the search space that have good chance of finding a solution
- Long-term memory
  - Diversification rules pointing to new non-explored areas

Ex: Taboo search and TSP

- Efficient graph structure exploration – ejection chain method
- Simple:
  - Random (or nearest neighbor) initial solution
  - Random swaps of two cities
  - Take better
  - Remember good visited solutions in taboo list

Scatter search

- The motivation was to improve global search by focusing on diversity of individuals
- Algorithm:
  - Generate initial population P(0)
  - Select reference set R(0) as subset of P(0)
- Cycle in time :
  - Generate new candidate solutions P(t) by arithmetic crossover of individuals from R(t-1)
  - Apply local changes (mutation, hill-climbing, taboo search) to P(t)
  - Update R(t) by (some) members of P(t)

Reference set

- Can be created and updated in various ways
- Criterium to be in R is typically some combination of good fitness and diversity (members of R have to be good and not similar to each other)
- maximize minimal distances of newly added x from the rest of R
- Can be updated:
  - incrementally (1 or more individuals per population),
  - Or completely renewed every population,
  - Or created in the inner cycle, new individuals are already members of R and recombination is performed with them.

Diferential evolution

- **Inicialization**: random
- **Mutation**: „shift“ based on other individuals
- **Crossover**: uniform
- **Parental selection**: all indivs, very fair
- **Environmental selection**: comparison of parent, replacing if the offspring is better
    - INITIALIZATION->MUTATION->CROSSOVER->SELECTION

Algorithm

- Initialize all agents x with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
- For each agent in the population do:
  - Pick three agents a,b,c from the population at random, they must be distinct from each other as well as from agent
  - Compute the agent's potentially new position y as follows:
    - y = a + F* (b-c)
  - If f(y)>f(x) then replace x with y.
- Pick the agent from the population that has the highest fitness or lowest cost and return it as the best found candidate solution.

Mutation

- Each individual undergoes mutation, crossover and (environmental) selection
- For indiv $\boldsymbol{x}_{i,p}$ choose three different indivs $\boldsymbol{x}_{a,p}$, $\boldsymbol{x}_{b,p}$, $\boldsymbol{x}_{c,p}$
- 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 mutation parameter, real number from <0;2>

Crossover

- Uniform crossover of original individual with donor
- Parameter *C* definies a probability of change
- In the new offspring, at least one part of donor
- Trial 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 random number from <0;1>
- I_{rand} is a random integer from <1;2; ... ; D>

Selection

- Compare fitness of $\boldsymbol{x}$ and $\boldsymbol{v}$, and take the better one:
  - $\boldsymbol{x}_{i,p+1} = \boldsymbol{u}_{i,p+1}$; iff $f(\boldsymbol{u}_{i,p+1}) <= f(\boldsymbol{x}_{i,p})$
  - $\boldsymbol{x}_{i,p+1} = \boldsymbol{x}_{i,p}$; else
  - for $i=1,2, \ldots , N$
- Mutation, crossover and selection is repeated until some termination criterion is satisfied, but you saw that coming