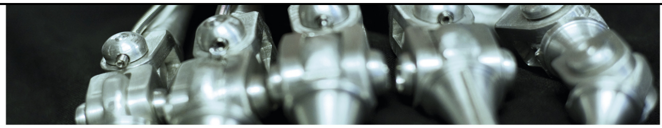


UiO : Department of Informatics
University of Oslo

INF3490 - Biologically inspired computing

Lecture 3: Eiben and Smith, chapter 5-6

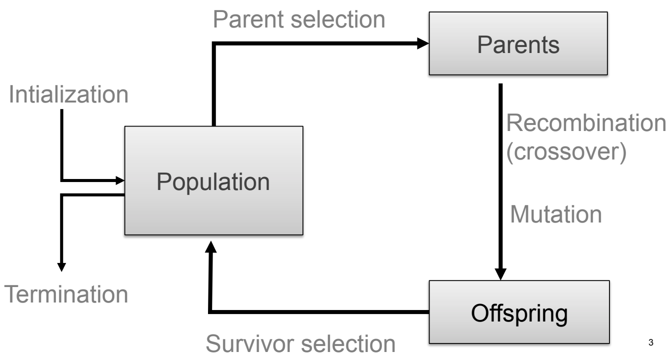
Evolutionary Algorithms - Population management and popular algorithms

Jim Tørresen

Chapter 5: Fitness, Selection and Population Management

- Selection is second fundamental force for evolutionary systems
- Components exist of:
  - Population management models
  - Selection operators
  - Preserving diversity

Scheme of an EA: General scheme of EAs



Population Management Models: Introduction

- Two different population management models exist:
  - **Generational model**
    - each individual survives for exactly one generation
    - the entire set of parents is replaced by the offspring
  - **Steady-state model**
    - one offspring is generated per generation
    - one member of population replaced
- Generation Gap
  - The proportion of the population replaced
  - Parameter = 1.0 for G-GA, = 1/pop_size for SS-GA

Population Management Models: Fitness based competition

- Selection can occur in two places:
  - Selection from current generation to take part in mating (**parent selection**)
  - Selection from parents + offspring to go into next generation (**survivor selection**)
- Selection operators work on whole individual
  - i.e. they are representation-independent !
- **Selection pressure**: As selection pressure increases, fitter solutions are more likely to survive, or be chosen as parents

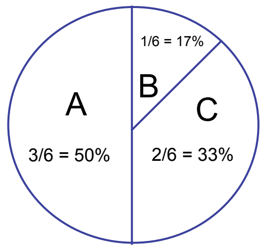
Parent Selection: Fitness-Proportionate Selection

Example: roulette wheel selection

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



Stochastic universal sampling (SUS)
Select multiple individuals by making **one** spin of the wheel with **a number of equally spaced arms**

Parent Selection: Fitness-Proportionate Selection (FPS)

- Probability for individual i to be selected for mating in a population size μ with FPS is

$$P_{FPS}(i) = f_i \Big/ \sum_{j=1}^{\mu} f_j$$

- Problems include
  - One highly fit member can rapidly take over if rest of population is much less fit: **Premature Convergence**
  - At end of runs when fitnesses are similar, loss of selection pressure
- **Scaling** can fix the last problem by:
  - **Windowing**: $f'(i) = f(i) - \beta^t$

    where β is worst fitness in this (last n) generations
  - **Sigma Scaling**: $f'(i) = \max(f(i) - (\overline{f} - c \bullet \sigma_f), 0)$

    where c is a constant, usually 2.0

Parent Selection: Rank-based Selection

- Attempt to remove problems of FPS by basing selection probabilities on ***relative* rather than *absolute* fitness**
- **Rank population** according to fitness and then base selection probabilities on rank (fittest has rank μ*-1* and worst rank 0)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

UiO : Department of Informatics
University of Oslo

Rank-based Selection: Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parameterised by factor *s:* $1 < s \leq 2$
  - measures advantage of best individual
- Simple 3 member example

Individual	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	0	0.1	0	0.167
B	4	1	0.4	0.33	0.33
C	5	2	0.5	0.67	0.5
Sum	10		1.0	1.0	1.0

UiO : Department of Informatics
University of Oslo

Rank-based selection: Exponential Ranking

$$P_{\exp-rank}(i) = \frac{1-e^{-i}}{c}$$

- Linear Ranking is limited in selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor *c* according to population size

Sample mating pool from the selection probability distribution (roulette wheel, stochastic universal sampling)

UiO : Department of Informatics
University of Oslo

Parent Selection: Tournament Selection (1/2)

- All methods above rely on global population statistics
  - Could be a bottleneck esp. on parallel machines, very large population
  - Relies on presence of external fitness function which might not exist: e.g. evolving game players
- Idea for a procedure using only local fitness information:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals

UiO : Department of Informatics
University of Oslo

Parent Selection: Tournament Selection (2/2)

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*

Parent Selection: Uniform

$$P_{uniform}(i) = \frac{1}{\mu}$$

- Parents are selected by uniform random distribution whenever an operator needs one/ some
- Uniform parent selection is unbiased - every individual has the **same probability** to be selected

Survivor Selection

- Managing the process of reducing the working memory of the EA from a set of μ parents and λ offspring to a set of μ individuals **forming the next generation**
- Survivor selection can be divided into two approaches:
  - **Age-Based Selection**
    - Fitness is not taken into account
    - In SS-GA can implement as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - **Fitness-Based Replacement**

Fitness-based replacement (1/2)

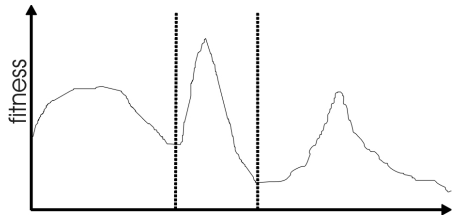
- Elitism
  - Always **keep** at least one copy of **the fittest solution** so far
  - Widely used in both population models (GGA, SSGA)
- GENITOR: a.k.a. “delete-worst”
  - Rapid takeover: use with large populations or “no duplicates” policy
- Round-robin tournament (from EP)
  - P(t): μ parents, P’(t): μ offspring
  - Pairwise competitions in round-robin format:
    - Each solution x from P(t) ∪ P’(t) is **evaluated against q other** randomly chosen solutions
    - For each comparison, a "win" is assigned if x is better than its opponent
    - The μ solutions with the greatest number of wins are retained to be parents of the next generation
  - Parameter q allows tuning selection pressure
  - Typically *q* = 10

Fitness-based replacement (2/2) (from ES)

- **(μ,λ)-selection** (best candidates can be lost)
  - based on the set of **children only** (λ > μ)
  - choose the **best** μ offspring for next generation
- **(μ+λ)-selection** (elitist strategy)
  - based on the set of **parents and children**
  - choose the **best** μ offspring for next generation
- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
- λ ≈ 7 • μ is a traditionally good setting (decreasing over the last couple of years, λ ≈ 3 • μ seems more popular lately)

Multimodality

Most interesting problems have more than one locally optimal solution.



Multimodality: Genetic Drift

- Finite population with global mixing and selection eventually convergence around one optimum
- Why?
- Often might want to identify several possible peaks
- Sub-optimum can be more attractive

Approaches for Preserving Diversity: Introduction (1/2)

- Explicit vs implicit
- Implicit approaches:
  - Impose an equivalent of geographical separation
  - Impose an equivalent of speciation
- Explicit approaches
  - Make similar individuals compete for resources (fitness)
  - Make similar individuals compete with each other for survival

Approaches for Preserving Diversity: Introduction (1/2)

Different spaces:

- Genotype space
  - Set of representable solutions
- Phenotype space
  - The end result
  - Neighbourhood structure may bear little relation with genotype space
- Algorithmic space
  - Equivalent of the geographical space on which life on earth has evolved
  - Structuring the population into a number of sub-populations

Explicit Approaches for Preserving Diversity: Fitness Sharing (1/2)

- Restricts the number of individuals within a given niche by “sharing” their fitness, so as to allocate individuals to niches **in proportion to the niche fitness**
- need to set the size of the niche σ_{share} in either genotype or phenotype space
- run EA as normal but after each generation set

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))} \qquad sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

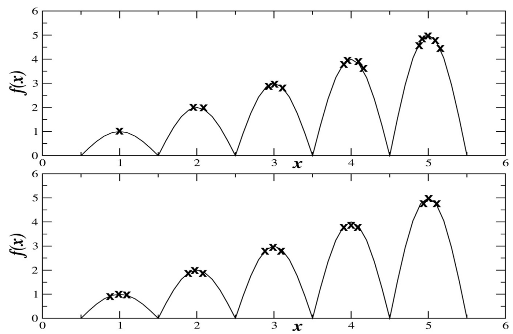
Explicit Approaches for Preserving Diversity: Fitness Sharing (2/2)

- Note: if we used sh(d) = 1 for d < σ_{share} then the sum that reduces the fitness would simply count the number of neighbours, i.e., individuals closer than σ_{share}
- This creates an advantage of being alone in the neighbourhood
- Using 1 – d/ σ_{share} instead of 1 implies that we count distant neighbours less

Explicit Approaches for Preserving Diversity: Crowding

- Attempts to distribute individuals **evenly** amongst niches
- relies on the assumption that offspring will tend to be close to parents
- uses a distance metric in ph/genotype space
- randomly shuffle and pair parents, produce 2 offspring
- Each offspring competes with their **nearest** parent for survival (using a distance measure)

Explicit Approaches for Preserving Diversity: Crowding or Fitness sharing?



Fitness Sharing

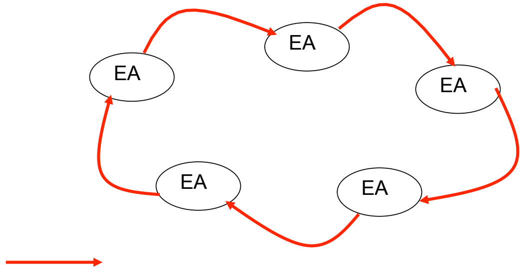
Crowding

Observe the number of individuals per niche

Implicit Approaches for Preserving Diversity: Automatic Speciation

- Either only mate with genotypically / phenotypically similar members or
- Add bits (tags) to problem representation
  - that are initially randomly set
  - subject to recombination and mutation
  - when selecting partner for recombination, only pick members with a good match

Implicit Approaches for Preserving Diversity: "Island" Model Parallel EAs



Periodic migration of individual solutions between populations

Implicit Approaches for Preserving Diversity: "Island" Model Parallel EAs

- Run multiple populations in parallel
- After a (usually fixed) number of generations (an ***Epoch***), exchange individuals with neighbours
- Repeat until ending criteria met
- Partially inspired by parallel/clustered systems

Chapter 6: Popular Evolutionary Algorithm Variants

Historical EA variants:

- Genetic Algorithms
- Evolution Strategies
- Evolutionary Programming
- Genetic Programming

Algorithm	**Chromosome Representation**	**Crossover**	**Mutation**
Genetic Algorithm (GA)	Array	X	X
Genetic Programming (GP)	Tree	X	X
Evolution Strategies (ES)	Array	(X)	X
Evolutionary Programming (EP)	No constraints	-	X

Genetic Algorithms: Overview Simple GA

- Developed: USA in the 1960's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete function optimization
  - benchmark for comparison with other algorithms
  - straightforward problems binary representation
- Features:
  - not too fast
  - missing new variants (elitsm, sus)
  - often modelled by theorists

Genetic Algorithms: Simple GA (SGA) summary

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional – implemented by Roulette Wheel
Survivor selection	Generational

Genetic Algorithms: SGA reproduction cycle

- **Select parents** for the mating pool (size of mating pool = population size)
- Shuffle the mating pool
- **Apply crossover** for each consecutive pair with probability p_c, otherwise copy parents
- **Apply mutation** for each offspring (bit-flip with probability p_m independently for each bit)
- **Replace the whole population** with the resulting offspring

Genetic Algorithms: An example after Goldberg '89

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g., 01101 ↔ 13
  - Population size: 4
  - 1-point x-over, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

UiO : Department of Informatics
University of Oslo

X² example: Selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

UiO : Department of Informatics
University of Oslo

X² example: Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

UiO : Department of Informatics
University of Oslo

X² example: Mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

UiO : Department of Informatics
University of Oslo

Genetic Algorithms: The simple GA

- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.,
  - Representation is too restrictive
  - Mutation & crossover operators only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model can be improved with explicit survivor selection

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Quick overview

- Developed: Germany in the 1960's
- Early names: I. Rechenberg, H.-P. Schwefel
- Typically applied to:
  - numerical optimisation
- Attributed features:
  - fast
  - good optimizer for real-valued optimisation
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard

UiO : Department of Informatics
University of Oslo

Evolution Strategies: ES summary

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Example (1+1) ES

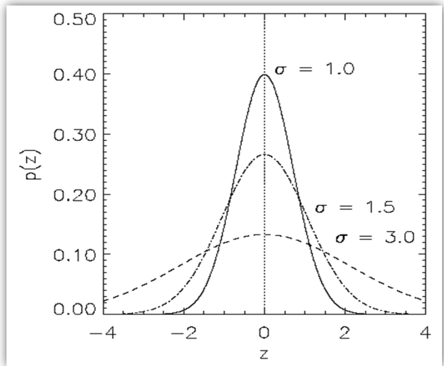
- Task: minimimise $f : R^n \rightarrow R$
- Algorithm: "two-membered ES" using
  - Vectors from R^n directly as chromosomes
  - Population size 1
  - Only mutation creating one child
  - Greedy selection

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Introductory example: mutation mechanism

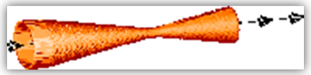
- z values drawn from normal distribution $N(\xi,\sigma)$
  - mean ξ is set to 0
  - variation σ is called **mutation step size**
- σ is varied on the fly by the "1/5 success rule":
- This rule resets σ after every k iterations by
  - $\sigma = \sigma / c$ if $p_s > 1/5$
  - $\sigma = \sigma \cdot c$ if $p_s < 1/5$
  - $\sigma = \sigma$ if $p_s = 1/5$
- where p_s is the % of successful mutations, $0.8 \leq c \leq 1$

Evolution Strategies: Illustration of normal distribution



Another historical example: the jet nozzle experiment

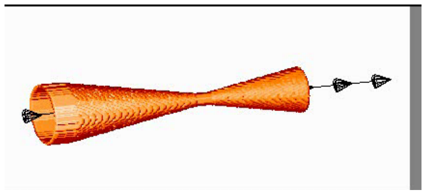
Task: to optimize the shape of a jet nozzle
Approach: random mutations to shape + selection



Initial shape

Final shape

The famous jet nozzle experiment (movie)



Evolution Strategies: Representation

- Chromosomes consist of three parts:
  - Object variables: $x_1,\dots,x_n$
  - Strategy parameters (ref. sec 4.4.2):
    - Mutation step sizes: $\sigma_1,\dots,\sigma_{n_\sigma}$
    - Rotation angles: $\alpha_1,\dots,\alpha_{n_\alpha}$
- Not every component is always present
- Full size: $\langle x_1,\dots,x_n, \sigma_1,\dots,\sigma_n, \alpha_1,\dots, \alpha_k \rangle$
  where k = n(n-1)/2 (no. of i,j pairs)

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Recombination

- Creates one child
- Acts per variable / position by either
  - Averaging parental values, or
  - Selecting one of the parental values
- From two or more parents by either:
  - Using two selected parents to make a child
  - Selecting two parents for each position

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Names of recombinations

	Two fixed parents	**Two parents selected for each i**
$\mathbf{z_i = (x_i + y_i)/2}$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Parent selection

- Parents are selected by uniform random distribution whenever an operator needs one/ some
- Thus: ES parent selection is unbiased - every individual has the **same probability** to be selected

UiO : Department of Informatics
University of Oslo

Evolution Strategies: Prerequisites for self-adaptation

- $\mu > 1$ to carry different strategies
- $\lambda > \mu$ to generate offspring surplus
- (μ,λ)-selection to get rid of misadapted σ's
- Mixing strategy parameters by (intermediary) recombination on them

Evolutionary Programming: Quick overview

- Developed: USA in the 1960's
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: prediction by finite state machines
  - contemporary EP: (numerical) optimization
- Attributed features:
  - very open framework: any representation and mutation op's OK
  - crossbred with ES (contemporary EP)
  - consequently: hard to say what "standard" EP is
- Special:
  - **no recombination**
  - self-adaptation of parameters standard (contemporary EP)

Evolutionary Programming: Technical summary tableau

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic (each parent one offspring)
Survivor selection	Probabilistic ($\mu+\mu$)

Evolutionary Programming: Historical EP perspective

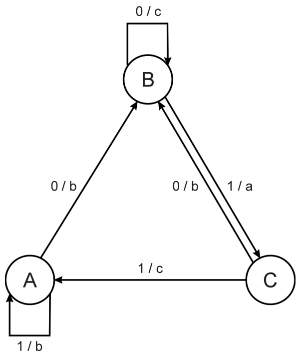
- EP aimed at achieving intelligence
- Intelligence was viewed as adaptive behaviour
- Prediction of the environment was considered a prerequisite to adaptive behaviour
- Thus: capability to predict is key to intelligence

Evolutionary Programming: Prediction by finite state machines

- Finite state machine (FSM):
  - States S
  - Inputs I
  - Outputs O
  - Transition function $\delta : S \times I \rightarrow S \times O$
  - Transforms input stream into output stream
- Can be used for predictions, e.g. to predict next input symbol in a sequence

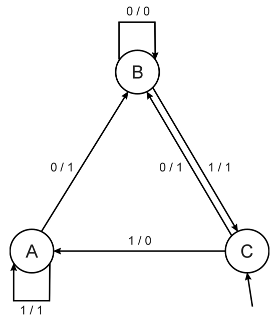
Evolutionary Programming: FSM example

- Consider the FSM with:
  - S = {A, B, C}
  - I = {0, 1}
  - O = {a, b, c}
  - δ given by a diagram



Evolutionary Programming: FSM as predictor

- Consider the following FSM
- Task: predict next input
- Quality: % of $in_{(i+1)} = out_i$
- Given initial state C
- Input sequence 011101
- Leads to output 110111
- Quality: 3 out of 5



Evolutionary Programming: Modern EP

- No predefined representation in general
- Thus: no predefined mutation (must match representation)
- Often applies self-adaptation of mutation parameters

Evolutionary Programming: Representation

- For continuous parameter optimisation
- Chromosomes consist of two parts:
  - Object variables: $x_1,\ldots,x_n$
  - Mutation step sizes: $\sigma_1,\ldots,\sigma_n$
- Full size: $\langle\, x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \,\rangle$

UiO : Department of Informatics
University of Oslo

Evolutionary Programming: Mutation

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \rangle$
- $\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1))$
- $x_i' = x_i + \sigma_i' \cdot N_i(0,1)$
- $\alpha \approx 0.2$
- boundary rule: $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$
- Other variants proposed & tried:
  - Using variance instead of standard deviation
  - Mutate σ-last
  - Other distributions, e.g, Cauchy instead of Gaussian

57

UiO : Department of Informatics
University of Oslo

Evolutionary Programming: Recombination

- None
- Rationale: one point in the search space stands for a species, not for an individual and there can be no crossover between species
- Much historical debate “mutation vs. crossover”

58

UiO : Department of Informatics
University of Oslo

Evolutionary Programming: Parent selection

- Each individual creates one child by mutation
- Thus:
  - Deterministic
  - Not biased by fitness

59

UiO : Department of Informatics
University of Oslo

Evolutionary Programming: Evolving checkers player (Fogel’02) (1/2)

- Neural nets for evaluating future values of moves are evolved
- NNs have fixed structure with 5046 weights, these are evolved + one weight for “kings”
- Representation:
  - vector of 5046 real numbers for object variables (weights)
  - vector of 5046 real numbers for σ‘s
- Population size 15

Evolutionary Programming: Evolving checkers player (Fogel'02) (2/2)

- Tournament size q = 5
- Programs (with NN inside) play against other programs, no human trainer or hard-wired intelligence
- After 840 generation (6 months!) best strategy was tested against humans via Internet
- Program earned "expert class" ranking outperforming 99.61% of all rated players

Genetic Programming: Quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary

Genetic Programming: Summary

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Genetic Programming: Example credit scoring (1/3)

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches historical data

ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

UiO : Department of Informatics
University of Oslo

Genetic Programming: Example credit scoring (2/3)

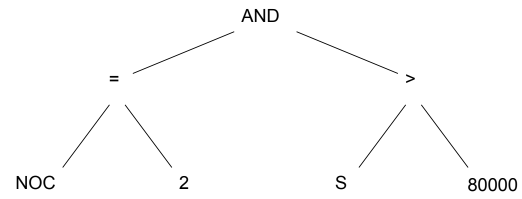
- A possible model:
- IF **(NOC = 2) AND (S > 80000)** THEN **good** ELSE **bad**
- In general:
- IF **formula** THEN **good** ELSE **bad**
- Only unknown is the right formula, hence
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for

UiO : Department of Informatics
University of Oslo

Genetic Programming: Example credit scoring (3/3)

IF **(NOC = 2) AND (S > 80000)** THEN **good** ELSE **bad**

can be represented by the following tree



UiO : Department of Informatics
University of Oslo

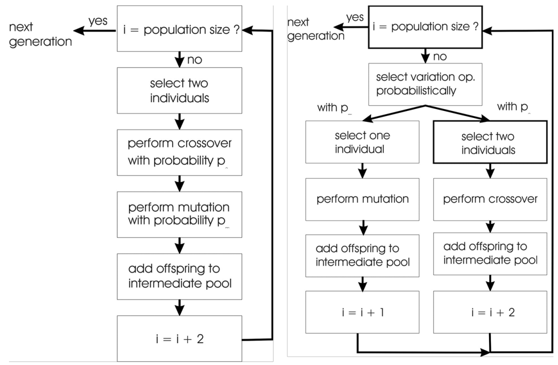
Genetic Programming: Offspring creation scheme

Compare

- GA scheme using crossover AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically)

UiO : Department of Informatics
University of Oslo

Genetic Programming: GA vs GP



GA flowchart GP flowchart

UiO : Department of Informatics
University of Oslo

Genetic Programming: Selection

- Parent selection typically fitness proportionate
- Over-selection in very large populations
  - rank population by fitness and divide it into two groups:
  - group 1: best x% of population, group 2 other (100-x)%
  - 80% of selection operations chooses from group 1, 20% from group 2
  - for pop. size = 1000, 2000, 4000, 8000 x = 32%, 16%, 8%, 4%
  - motivation: to increase efficiency, %'s come from rule of thumb
- Survivor selection:
  - Typical: generational scheme (thus none)
  - Recently steady-state is becoming popular for its elitism

UiO : Department of Informatics
University of Oslo

Genetic Programming: Initialisation

- Maximum initial depth of trees D_{max} is set
- Full method (each branch has depth = D_{max}):
  - nodes at depth d < D_{max} randomly chosen from **function set F**
  - nodes at depth d = D_{max} randomly chosen from **terminal set T**
- Grow method (each branch has depth ≤ D_{max}):
  - nodes at depth d < D_{max} randomly chosen from F ∪ T
  - nodes at depth d = D_{max} randomly chosen from T
- Common GP initialisation: ramped half-and-half, where grow & full method each deliver half of initial population

UiO : Department of Informatics
University of Oslo

Genetic Programming: Bloat

- Bloat = "survival of the fattest", i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver "too big" children
  - Parsimony pressure: penalty for being oversized

UiO : Department of Informatics
University of Oslo

Summary: The standard EA variants

Name	Representation	Crossover	Mutation	Parent selection	Survivor selection	Specialty
Genetic Algorithm	Usually fixed-length vector	Any or none	Any	Any	Any	None
Evolution Strategies	Real-valued vector	Discrete or intermediate recombination	Gaussian	Random draw	Best N	Strategy parameters
Evolutionary Programming	Real-valued vector	None	Gaussian	One child each	Tournament	Strategy parameters
Genetic Programming	Tree	Swap sub-tree	Replace sub-tree	Usually fitness proportional	Generational replacement	None