10 Popular Evolutionary Algorithm Variants

Popular Evolutionary Algorithm Variants

- we're going to take a tour of these main EA variants:
  - Genetic Algorithms (GA and SGA)
  - Evolution Strategies (ES)
  - Evolutionary Programming (EP)
  - Genetic Programming (GP)
- and later on in the course we'll look more recent variants, such as:
  - Particle Swarm Optimisation (PSO)
  - Ant Colony Optimisation (ACO)

Genetic Algorithms: Quick Overview

- developed in 1960s by John Holland
- initially conceived as a means of studying adaptive behaviour
- typically applied to:
    - discrete function optimization
    - benchmarking new algorithms
    - straightforward problems that allow binary representation
- have been lots of variants developed since
    - as we've seen
- so Holland's GA is now known as the Simple Genetic Algorithm (SGA)

Simple Genetic Algorithm (SGA): Technical Summary Tableau

representation	bit-strings
recombination	1-point crossover
mutation	bit flip
parent selection	fitness proportional – implemented by Roulette Wheel
survivor selection	generational

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

SGA Example

from Goldberg 1989:

- simple problem: max x^2 over $\{0,1,\ldots,31\}$
- GA approach:
  - representation: binary code, e.g., $01101 \leftrightarrow 13$
  - population size: 4
  - 1-point crossover, bitwise mutation
  - roulette wheel selection
  - random initialisation
- let's see one generational cycle done by hand…

X^2 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

X^2 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

X^2 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

The Simple GA

- has been subject of many (early) studies
- still often used as benchmark for novel GAs
- shows many shortcomings, such as:
  - representation is too restrictive
  - mutation and crossover operators are only applicable for bit-string and integer representations
  - selection mechanism is sensitive for converging populations with close fitness values
  - generational population model can be improved with explicit survivor selection

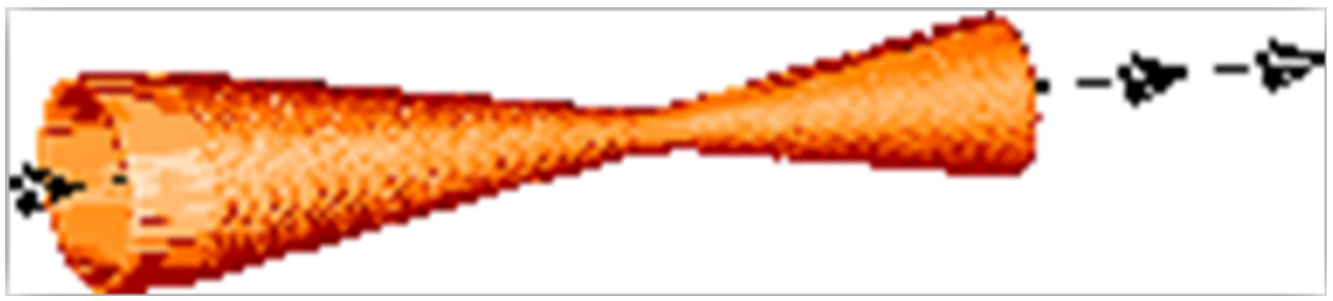
Evolution Strategies: Quick Overview

- developed by Rechenberg & Schwefel
- typically applied to:
    - numerical optimisation
- attributed features:
    - fast
    - good optimizer for real-valued optimisation
- special:
    - self-adaptation of mutation parameters is standard

An Historical Example:
The Jet Nozzle Experiment

- task:
  - optimise the shape of a jet nozzle
- approach:
  - random mutations to shape, with selection

before:



after:

Evolution Strategies: Technical Summary Tableau

representation	real-valued vectors
recombination	discrete or intermediary
mutation	Gaussian (Normal) pertubation
parent selection	uniform random
survivor selection	(μ,λ) or (μ+λ)

Evolution Strategies: (1+1) Example

- a very simple early version
- task: maximise $f : \mathbb{R}^n \to \mathbb{R}$
- algorithm is a 'two-membered ES':
  - uses vectors from $\mathbb{R}^n$ as genotypes
  - population size of 1
  - only uses mutation, creating one child
  - 'greedy' selection
    - accept offspring if fitter than parent
- (1,1) version always replaces the parent with the child

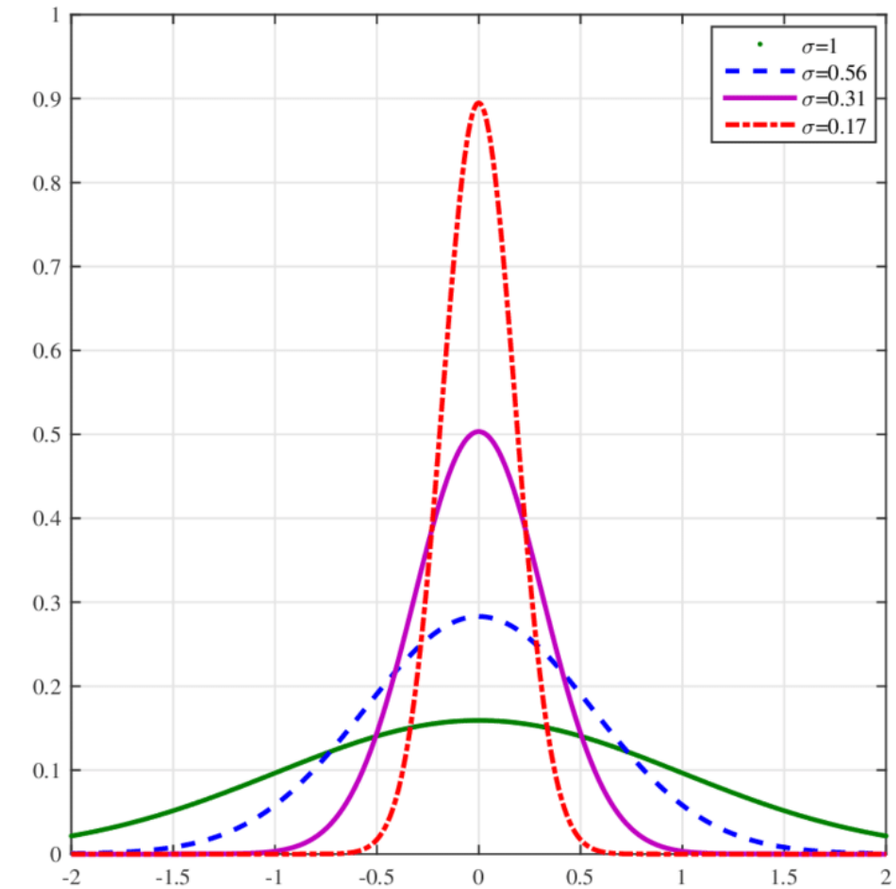
Evolution Strategies: Representation

- chromosomes consist of three parts:
  - object variables: $x_1, \ldots, x_n$
  - strategy parameters:
    - mutation step sizes: $\sigma_1, \ldots, \sigma_n$
    - rotation angles: $\alpha_1, \ldots, \alpha_n$ (for correlated mutation)
- not every component is always present
- full size genotype: $< x_1, \ldots, x_n, \ \sigma_1, \ldots, \sigma_n \ , \ \alpha_1, \ldots, \alpha_k >$

  where $k = n(n-1)/2$ (number of i, j pairs)

Evolution Strategies: Example: Mutation Mechanism

- random numbers drawn from normal distribution N(0,σ)
  - where 0 is the mean
  - and σ is the mutation step size
- σ is varied using the 1/5 success rule:
- for some k, every k generations, the step-size should:
  - increase if 'too many' steps are successful (>20%)
    - because this indicates that the search is too local
  - decrease if 'too few' steps are successful (<20%)
    - because this indicates that the step size is 'too large'
  - and stay the same if exactly 20% of steps are successful



Evolution Strategies: Recombination

- creates one child
- acts per gene position by either:
    - averaging the parental values, or
    - selecting one of the parental values
- derives values from two or more parents, either:
    - uses the same two parents for all positions
    - selects two different parents for each position

Evolution Strategies: Names of Recombinations

	two fixed parents	two parents selected for each `i`
$z_i = (x_i + y_i)/2$	local intermediary	global intermediary
z_i is x_i or y_i chosen randomly	local discrete	global discrete

Evolution Strategies: Parent Selection

- parents are selected by uniform random distribution
- so ES parent selection is unbiased
- every individual has the same probability of being selected

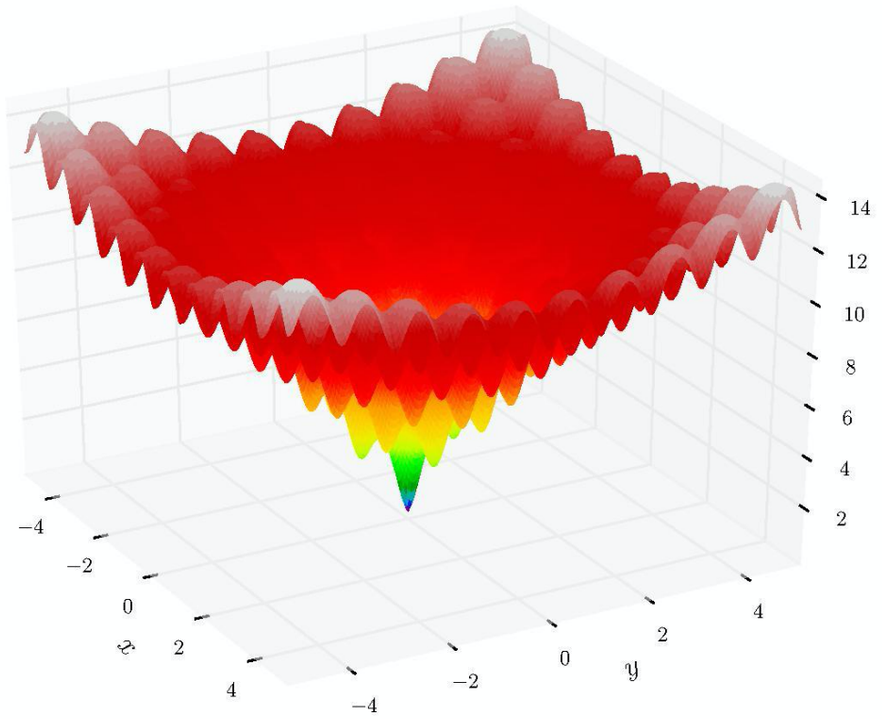
Evolution Strategies: Survivor Selection

- generally (μ, λ) is preferred to $(\mu + \lambda)$ because:
  - (μ,λ) discards all parents
    - so can in principle leave (small) local optima
  - if the fitness function changes over time, $(\mu + \lambda)$ selection preserves outdated solutions
    - so is less able to follow the moving optimum
  - $(\mu + \lambda)$ selection hinders self-adaptation, because misadapted strategy parameters may survive for a relatively large number of generations
- selective pressure in evolution strategies is very high because λ is typically much higher than μ
- typically an ES is a more aggressive optimizer than a (simple) GA

ES Example: The Ackley Function

evolution strategy:

- representation:
  - $-30 < x_i < 30$
- $(30,200)$ survivor selection
  - $(\mu=30,\lambda=200)$
- termination:
  - after $200,000$ fitness evaluations
- results:
  - average best solution is 7.48×10^{-8}
  - which is very good!



$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right)$$

$$-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

Evolutionary Programming: Quick Overview

- originally developed by Fogel et al in the 1960s
- why?
    - to simulate evolution as a learning process with the aim of generating artificial intelligence
- intelligence was viewed as adaptive behaviour
- so capability to predict is considered key to intelligence

Evolutionary Programming: Quick Overview

- typically applied to:
    - traditional EP: prediction by finite state machines
    - contemporary EP: (numerical) optimization
- attributed features:
    - very open framework:
        - any representation and mutation operators are OK
    - contemporary EP are ‘crossbred’ with ES techniques
        - consequently it’s hard to say what ‘standard’ EP is
- special:
    - no recombination
    - self-adaptation of parameters standard (contemporary EP)

Evolutionary Programming: Technical Summary Tableau

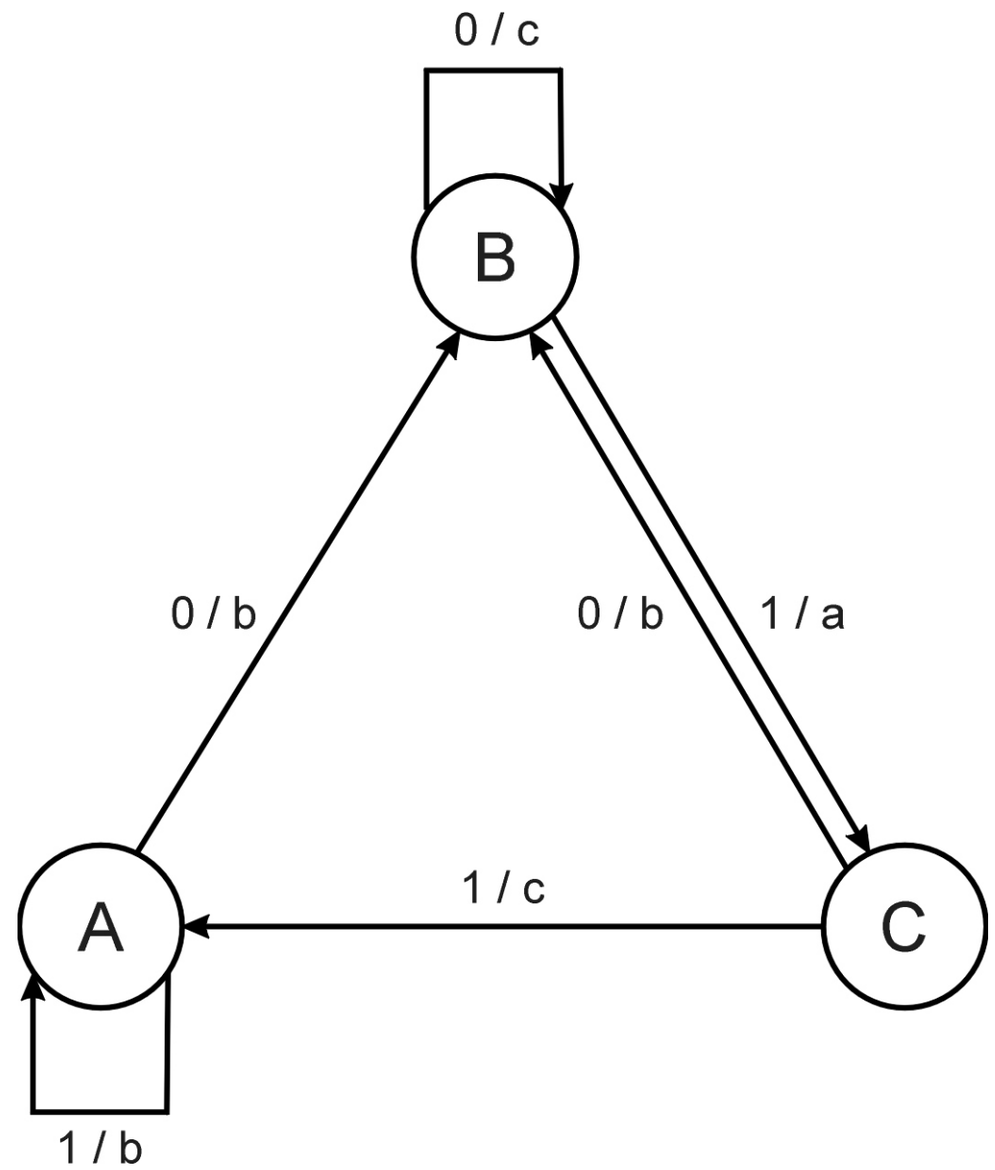
representation	real-valued vectors
recombination	none
mutation	Gaussian (Normal) pertubation
parent selection	deterministic (each parent produces one offspring)
survivor selection	probabilistic (μ+μ)

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, such as predicting the 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}`
  - `d`, given by a diagram:

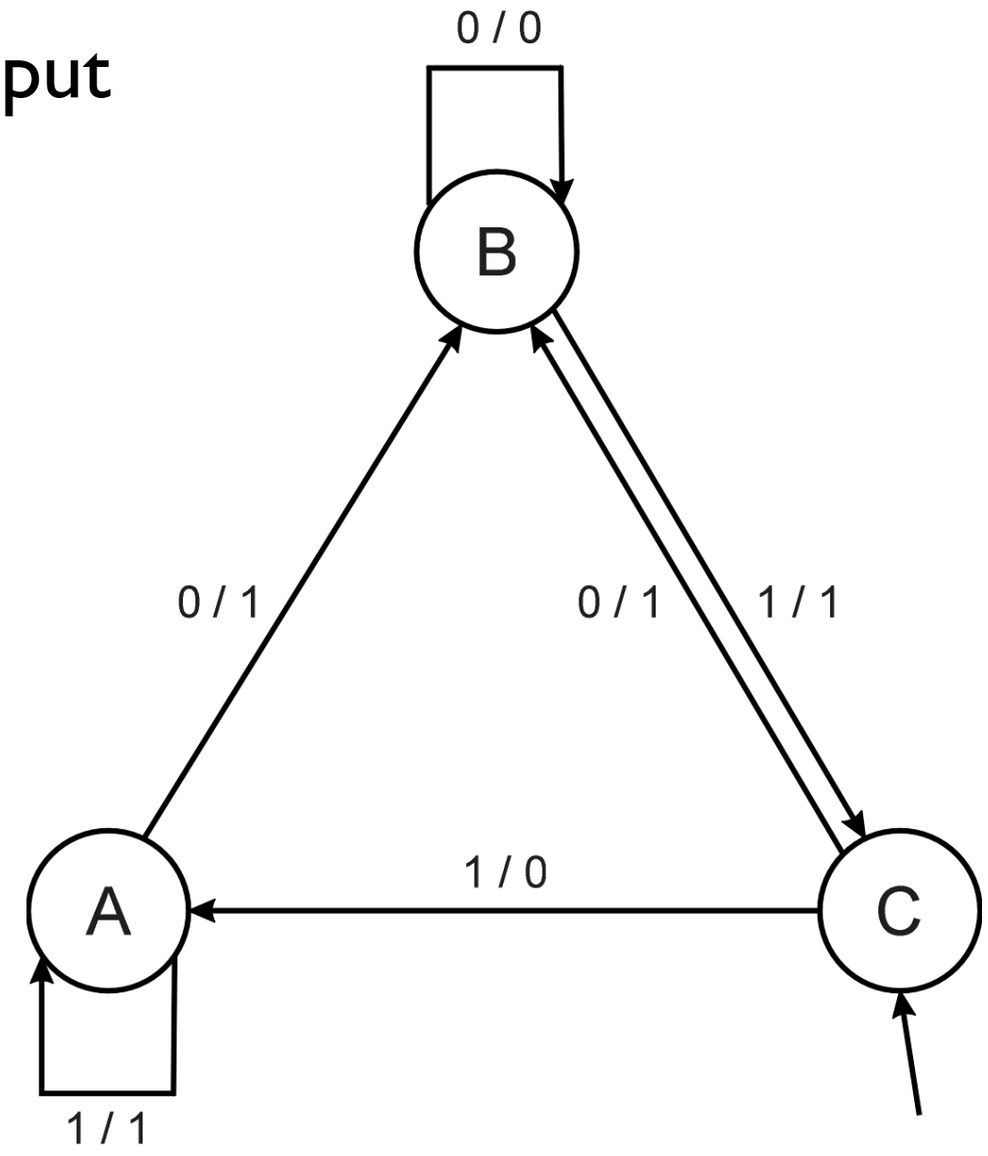


Evolutionary Programming: FSM As Predictor

- consider an FSM to predict next input
- quality: % of $in_{(i+1)} = out_i$
- given initial state C
- example:

i	1	2	3	4	5	6
input	0	1	1	1	0	1
output	1	1	0	1	1	1

quality: 60% (3 out of 5)



Evolutionary Programming: Evolving FMS to Predict Primes

- `P(n) = 1` if `n` is prime, `0` otherwise
- `I = ℕ = {1,2,3,…, n, …}`
- `O = {0,1}`
- correct prediction: $out_i = P(in_{(i+1)})$
- fitness function:
  - 1 point for correct prediction of next input
  - 0 point for incorrect prediction
  - penalty for ‘too many’ states

Evolutionary Programming: Evolving FMS to Predict Primes

- parent selection: each FSM is mutated once
- mutation operators (one selected randomly):
    - change an output symbol
    - change a state transition (i.e. redirect edge)
    - add a state
    - delete a state
    - change the initial state
- survivor selection: (μ+μ)
- results: overfitting, after 202 inputs best FSM had one state and both outputs were 0,
    - so it always predicted 'not prime'!
- *take away thought:* not perfect accuracy, but proof that simulated evolutionary process can create good solutions for intelligent task

Evolutionary Programming: Modern EP

- no predefined representation in general
- so 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 genotype: $< x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n >$

Evolutionary Programming: Mutation

- genotypes: $< x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n >$
- $\sigma_1' = \sigma_1 \bullet (1 + \alpha \bullet N(0,1))$
- $x_1' = x_1 + \sigma_1' \bullet N_i(0,1))$
- $\alpha \approx 0.2$
    - (so this is correlated stepwise mutation seen in previous slides)
- boundary rule: $\sigma' < \varepsilon \Rightarrow \sigma' = \varepsilon$
- other variants proposed and tried:
    - using variance instead of standard deviation
    - mutate σ-last
    - other distributions, such as Cauchy instead of Gaussian

Evolutionary Programming: Recombination

- none
- why?
- because EP has a different biological inspiration to GA and ES
- in EP one point in the search space stands for a species, not for an individual
- so there can be no crossover between species
- much historical debate about the benefits of 'mutation versus crossover'

Evolutionary Programming: Parent & Survivor Selection

- parent selection:
  - each individual creates one child by mutation
  - deterministic
    - not biased by fitness
- survivor selection:
  - parents and offspring populations are merged and compete in stochastic round-robin tournaments for survival

Evolutionary Programming: Evolving Checkers Players

- Fogel 2002:
- neural nets for evaluating future values of moves are evolved
- nets have fixed structure with 5046 weights
- these are evolved, plus one weight for 'kings'
- representation:
    - vector of 5046 real numbers for object variables (weights)
    - vector of 5046 real numbers for σs
- mutation:
    - correlated stepwise, σ-first
    - plus special mechanism for the kings' weight
- population size of 15

Evolutionary Programming: Evolving Checkers Players

- selection:
  - tournament size q = 5
  - programs (with nets inside) play against other programs
    - no human trainer or hard-wired intelligence
- after 840 generations (6 months computing time!) the best strategy was tested against humans via Internet
- program earned 'expert class' ranking, outperforming 99.61% of all rated players

Genetic Programming: Quick Overview

"programming of computers by means of natural selection" *or*

"automatic evolution of computer programs"

- developed in the 1990's
- 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: Technical Summary Tableau

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

- 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
…				

Genetic Programming: Example: Credit Scoring

- a possible model:

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

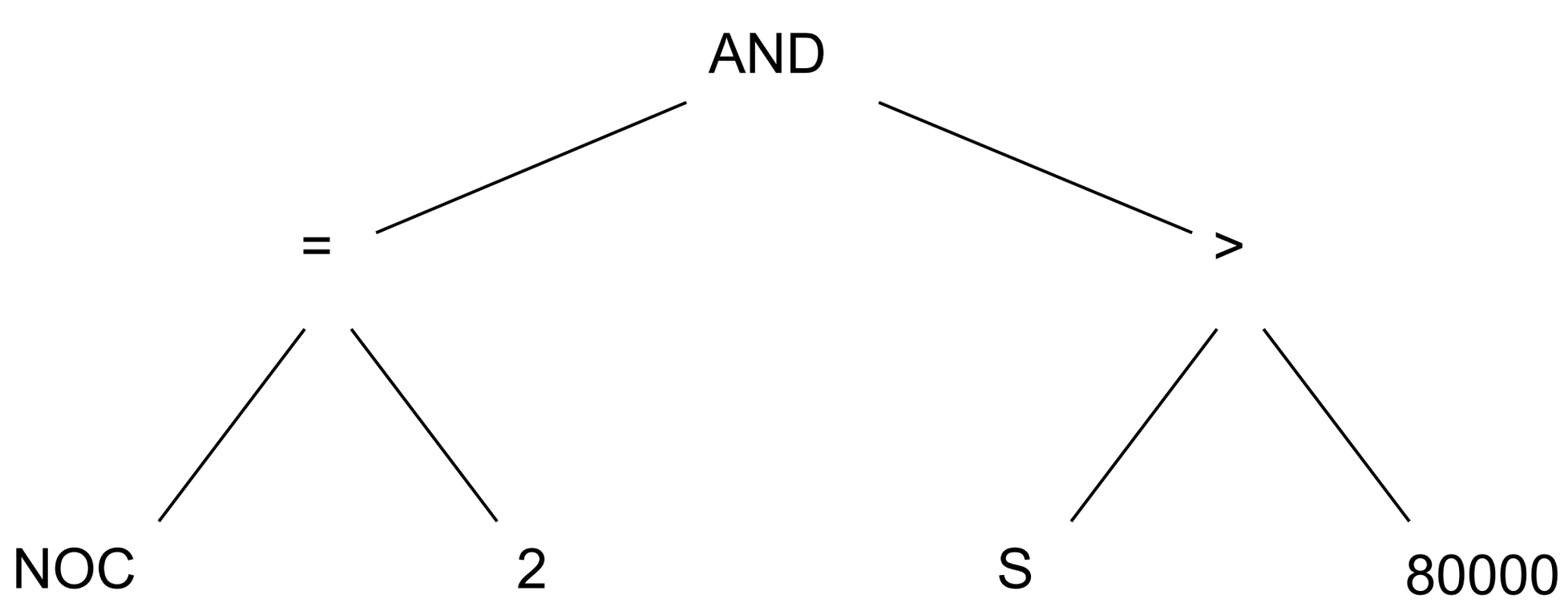
- in general we see that this takes the form:

```
IF formula THEN good ELSE bad
```

- the only unknown is the right formula, hence our search space (phenotypes) is the set of formulas
- the natural fitness of a formula is percentage of well classified (test) cases of the model it stands for

Genetic Programming: Example: Credit Scoring

`IF (NOC = 2) AND (S > 80000) THEN good ELSE bad`
can be represented by the following tree:



Genetic Programming: Initialisation

- most common method is ramped half-and-half
- given sets of functions `F` and terminals `T`
- a maximum initial depth for trees is chosen, D_{max}
- each member of the initial population is created using one of two methods, each with equal probability:
- full method:
    - each branch has depth D_{max}
    - beginning at the root:
        - nodes at depth `d` < D_{max} are randomly chosen from function set `F`
        - nodes at depth `d` = D_{max} are randomly chosen from function set `T`
- grow method:
    - each branch has `depth` <= D_{max}
    - beginning at the root:
        - nodes at depth `d` < D_{max} are randomly chosen from function set `F U T`
        - nodes at depth `d` = D_{max} are randomly chosen from function set `T`

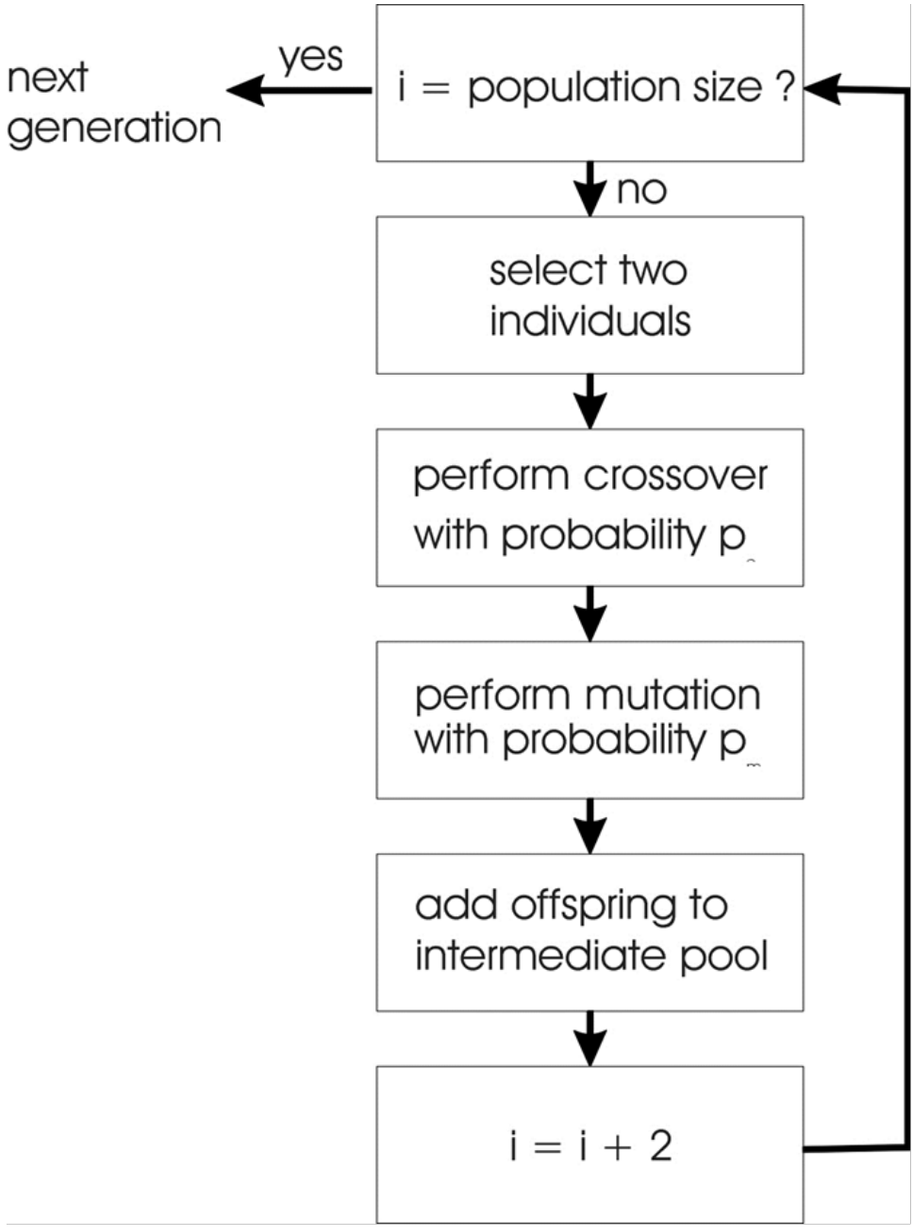
Genetic Programming: Mutation

- typically there is a low or zero mutation rate
- early on Koza advised using no mutation at all
- later Banzhaf et al advised a low rate of around 5%
- so very different from other EA variants
    - but why?
- the consensus is that crossover in GP has a large shuffling effect
- which acts as a kind of 'macromutation' operator

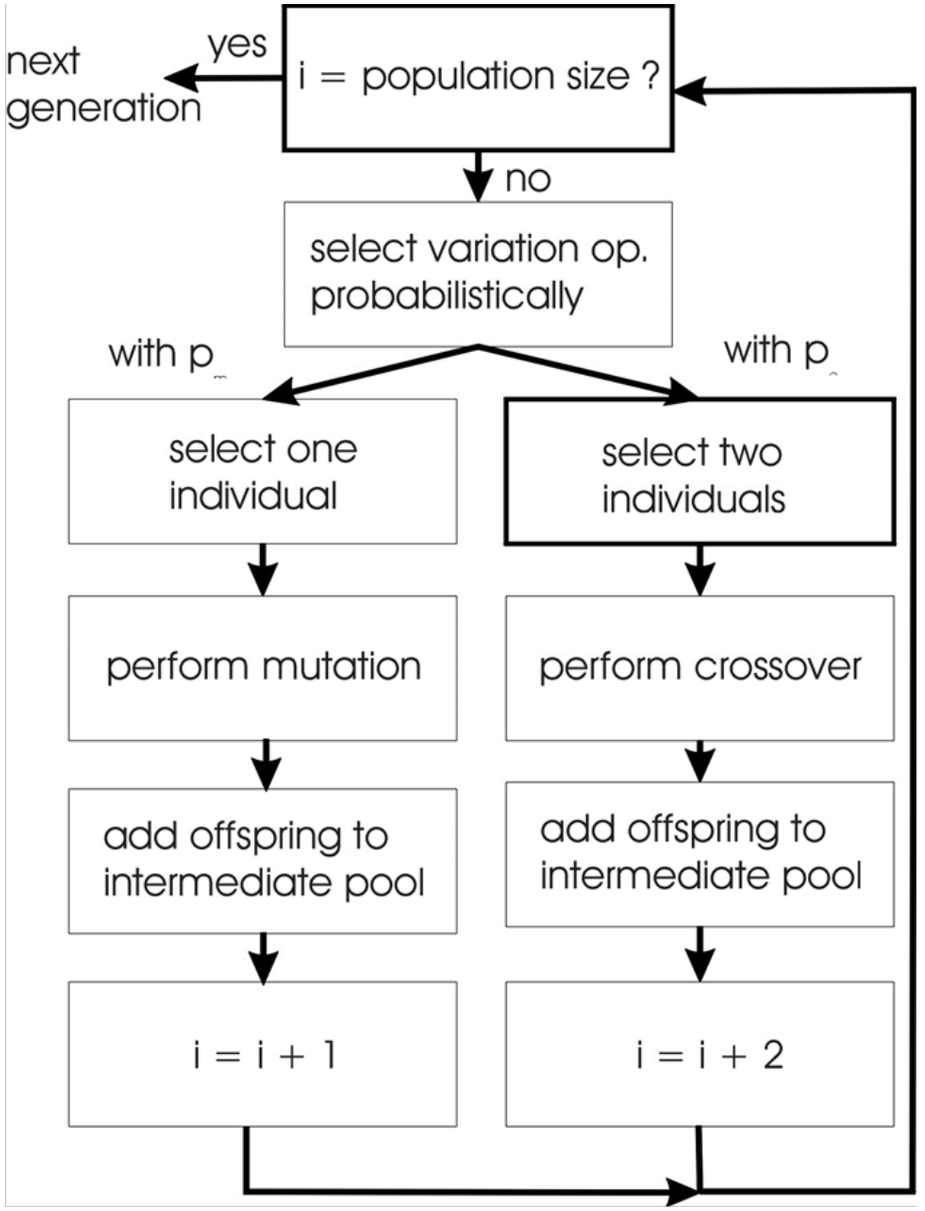
Genetic Programming: Offspring Creation Scheme

- let's compare:
  - GA scheme using crossover AND mutation
- with:
  - GP scheme using crossover OR mutation
  - (chosen probabilistically)

Genetic Programming: GA versus GP



GA flowchart



GP flowchart

Genetic Programming: Selection

- parent selection is typically fitness proportionate
- over-selection is often used with very large populations, to increase efficiency
    - rank population by fitness and divide it into two groups:
        - group 1: the best x% of population
        - group 2: the other (100-x)%
    - then:
        - 80% of selection operations choose from group 1
        - 20% choose from group 2
    - for population sizes 1000, 2000, 4000, 8000, good x values are 32%, 16%, 8%, 4%
- survivor selection:
    - typically: generational scheme (none)
    - recently: steady-state is becoming popular because of the elitism that it provides

Genetic Programming: Bloat

Bloat: "survival of the fattest"

- so tree sizes in the population are increasing over time
- ongoing research and debate about the reasons
- not much insight, other than "because it's possible they can, and so without any pressure for smaller sizes, they will!"
- so needs countermeasures, such as:
    - prohibiting variation operators that would deliver 'too big' children
        - but this could inhibit the search
    - parsimony pressure: a penalty for being oversized
        - more widely used

Genetic Programming: Example: Symbolic Regression

- given some points in $\mathbb{R}^n$, $(x_1, y_1), \ldots, (x_n, y_n)$
- find function $f(x) : \forall i = 1, \ldots, n : f(x_i) = y_i$
- possible GP solution:
  - representation:
    - F = {+, -, /, sin, cos}
    - T = R U {x}
  - fitness is the error
  - all operators standard
  - population size = 1000, ramped half-and-half initialisation
  - termination: n 'hits' or 50000 fitness evaluations reached
    - (where 'hit' is if $|\ f(x_i) - y_i\ | < 0.0001$)

Reading & References

- slides based on and adapted from, Chapter 6 (and slides) of Eiben & Smith's *Introduction to Evolutionary Computing*
- the book also mentions several other variants not covered here
- see the Resources section of Brightspace for wider reading
- Genetic Algorithms:
  - Kenneth De Jong, *Genetic algorithms are NOT function optimizers*, pp 5–18 in Foundations of Genetic Algorithms 2, Morgan Kaufmann, San Francisco, 1993
  - D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989
  - J.H. Holland, *Adaption in Natural and Artificial Systems*, MIT Press, 1992
  - M.Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996 (a good supplementary course text book!)

Reading & References

- Evolution Strategies:
  - T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996
  - T. Bäck, D.B. Fogel, and Z.Michalewicz, *Evolutionary Computation 1: Basic Algorithms and Operators*, Institute of Physics Publishing, 2000
  - T. Bäck, D.B. Fogel, and Z.Michalewicz, *Evolutionary Computation 2: Advanced Algorithms and Operators*, Institute of Physics Publishing, 2000
  - Hans-Georg Beyer, *The Theory of Evolution Strategies*, Springer, Berlin, 2001
  - H.-G. Beyer and H.-P. Schwefel, *Evolution strategies: A comprehensive introduction. Natural Computing,* 1(1):3–52, 2002
  - H.-P. Schwefel, *Evolution and Optimum Seeking*, Wiley, New York, 1995

Reading & References

- Evolutionary Programming:
  - L.J. Fogel, A.J. Owens, M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley, 1966
  - D.B. Fogel, *Evolutionary Computation*, IEEE Press, Piscataway, NJ, 1995
  - T. Bäck, G.Rudolph, H.-P.Schwefel, *Evolutionary programming and evolution strategies: Similarities and differences*, in Fogel, Atmar Eds. Proceedings of EP-93, pp. 11–22
  - D.B.~Fogel, *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann, San Francisco, 2002

Reading & References

- Evolutionary Programming:
  - J.R. Koza, *Genetic Programming*, MIT Press, 1992
  - J.R. Koza, *Genetic Programming II*, MIT Press, 1994
  - W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, 1998 (a good text if you ever pursue this type of work)
  - W.B. Langdon, *Genetic Programming* + *Data Structures* = *Automatic Programming!*, Kluwer, 1998
  - W.B. Langdon and R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, 2001