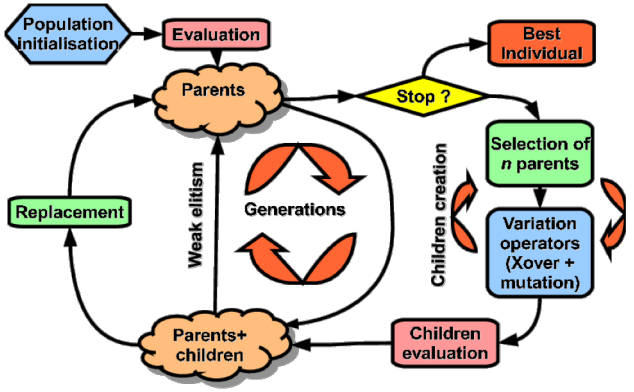
NTNU

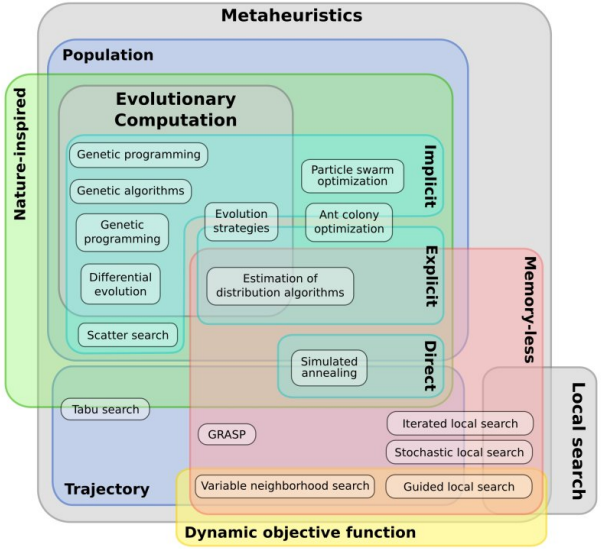
Norwegian University of Science and Technology

# Lecture 4

## Variants of Evolutionary Algorithms

Håken Jevne,
Kazi Ripon and Pauline Haddow



NTNU

# Outline

- Evolution Strategies
- Evolutionary Programming
- Genetic Programming
- Coevolution
- Hybridized EA

NTNU

# Outline

- **Evolution Strategies**
- Evolutionary Programming
- Genetic Programming
- Coevolution
- Hybridized EA

# Evolution Strategies (ES)

- Developed: Germany in the 1970's.
- Early names: I. Rechenberg, H.-P. Schwefel.
- Typically applied to:
  - numerical optimisation.
  - ***Continuous parameter optimization.***
- Attributed features:
  - Fast.
  - good optimizer for ***real-valued optimisation***.
  - relatively much theory.
- Special:
  - ***self-adaptation*** of (mutation) parameters standard.

# ES: early and modern version

- Early Evolutionary Strategy:
  - uses mutation only.
  - applied to real-valued continuous variables.
  - only a single solution was maintained .

- Later developments:
  - use crossover operators also.
  - also applies to discrete variables.
  - ES's are a population based approach.

# Major Characteristics

- Strong emphasis on mutation for creating offspring.
- Mutation is implemented through adding some random noise drawn from Gaussian Distribution.
- Mutation parameters are changed during a run of the algorithm.
- *Small changes have small effects*.

# Self-Adaptation of Strategy Parameters

- Some parameters of EA are varied during a run in a specific manner.
- Parameters are included in the chromosomes,
  - *And coevolve with the solutions*.
- Inherent in modern ES.
- **1/5 success rule:** resets mutation step size ($\sigma$) after every *k* iterations by

$$\sigma = \sigma / c \quad \text{if } p_s > 1/5$$

$$\sigma = \sigma \bullet c \quad \text{if } p_s < 1/5$$

$$\sigma = \sigma \quad \text{if } p_s = 1/5$$

where $p_s$ is the % of successful mutations, $0.817 \leq c \leq 1$

# Pseudocde

Set $t = 0$

Create initial point $x^t = \langle x_1^t, \ldots, x_n^t \rangle$

REPEAT UNTIL (*TERMIN.COND* satisfied) DO

Draw $z_i$ from a normal distr. for all $i = 1, \ldots, n$

$y_i^t = x_i^t + z_i$

IF $f(x^t) \geq f(y^t)$ THEN

$x^{t+1} = x^t$

ELSE

$x^{t+1} = y^t$

FI

Set $t = t+1$

OD

# Basic ES

In its simplest form, termed as a (1+1)-ES, one parent generates one offspring per generation by applying ***normally distributed*** mutation.

The (1+1)-evolution strategy can be implemented as follows:

**Step 1:** Choose the number of parameters *N* to represent the problem, and then determine a feasible range for each parameter:

$$\{x_{1min},\ x_{1max}\},\ \{x_{2min},\ x_{2max}\},\ \ldots,\ \{x_{Nmin},\ x_{Nmax}\}$$

Define a standard deviation for each parameter and the function to be optimized.

# Basic ES

**Step 2:** Randomly select an initial value for each parameter from the respective feasible range.

The set of these parameters will constitute the initial population of parent parameters:

$$x_1, x_2, \ldots, x_N$$

**Step 3:** Calculate the solution associated with the parent parameters:

$$X = f(x_1, x_2, \ldots, x_N)$$

# Basic ES

**Step 4: Mutation**

- Main mechanism: changing value by adding random noise drawn from normal (Gaussian) distribution.
  - $x'_i = x_i + N(0,\sigma)$
    - $N(0, \sigma)$ is a random Gaussian number with a mean of zero and standard deviations of σ.
- Key idea:
  - σ is part of the chromosome $\langle x_1,\ldots,x_n, \sigma \rangle$
  - σ is also mutated into σ' (see later how).
- *Thus: mutation step size $\sigma$ is coevolving with the solution x.*

# Basic ES

**Step 5:** Calculate the solution associated with the offspring parameters:

$$X' = f\left(x'_1, x'_2, \ldots, x'_N\right)$$

**Step 6:** Compare the solution associated with the offspring parameters with the one associated with the parent parameters.

**Step 7:** Go to Step 4, and repeat the process until a satisfactory solution is reached, or a specified number of generations is considered.

# Representation

- Full size: $\langle \underbrace{x_1, \ldots, x_n}_{\bar{x}}, \underbrace{\sigma_1, \ldots, \sigma_{n_\sigma}}_{\bar{\sigma}}, \underbrace{\alpha_1, \ldots, \alpha_{n_\alpha}}_{\bar{\alpha}} \rangle$
- Chromosomes consist of three parts:
  - Object variables: $x_1, \ldots, x_n$
  - Strategy parameters:
    - Mutation step sizes: $\sigma_1, \ldots, \sigma_{n\sigma}$
    - Rotation angles: $\alpha_1, \ldots, \alpha_{n\alpha}$
- Not every component is always present.

# Recombination (Modern ES)

- Creates one child.
- Acts per variable / position by either
  - Averaging parental values (Intermediate)
  - Selecting one of the parental values (Discrete)
- From two or more parents by either:
  - Using two selected parents to make a child (Local)
  - Selecting two parents for each position a new (Global)

| | 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 |

# Intermediate Recombination

- Often used to adapt the strategy parameters.
- Each child parameter is the mean value of the corresponding parent parameters.

(8, 12, 31, … ,5)  (2, 5, 23, … , 14)

(5, 8.5, 27, … , 9.5)

# Discrete Recombination

- Similar to crossover of genetic algorithms.
- Equal probability of receiving each parameter from each parent.

$$(8, 12, 31, \ldots, 5)\ (2, 5, 23, \ldots, 14)$$

$$(2, 12, 31, \ldots, 14)$$

# Parent Selection

- Unbiased of fitness function- every individual has the same probability to be selected.
  - Parents are selected randomly with uniform distribution from the population of $\mu$, individuals.
- Note that in **ES "parent" means the whole population member** (in GA's: parent is a selected member of the population to undergo variation).

# Survivor Selection

- Applied after creating $\lambda$ children from the $\mu$ parents by mutation and recombination.
- Deterministically chops off the “bad stuff”.
- Basis of selection is either:
  - The set of children only: $(\mu,\lambda)$-selection.
  - The set of parents and children: $(\mu+\lambda)$-selection.
- $(\mu+\lambda)$-selection is an elitist strategy.
- $(\mu,\lambda)$-selection can “forget”.

# Differences Between GA & ES

The differences, particularly in the early work include:

| | GA | ES |
|---|---|---|
| Population | Always population Based | Single |
| Representations | Binary | Real-Valued |
| Evolutionary Operators | Crossover and mutation | Only mutation |
| Selection mechanism | Biased on fitness | Deterministic |

Latterly, these differences have become eroded as ES have become population based approaches and have also been used for discrete representations.

NTNU

# Applications of ES

- Configuration of ventilation systems.
- Piping systems.
- Wing profiles and fuselage configurations.
- Optical cable system layout
- ……

NTNU

# Common Applications

**GA**

- More important to find optimal solution (GA's more likely to find global maximum; usually slower)
- Problem parameters can be represented as bit strings (computational problems)

**ES**

- "Good enough" solution acceptable (ES's usually faster; can readily find local maximum)
- Problem parameters are real numbers (engineering problems) – Early ES

NTNU

# Outline

- Evolution Strategies
- **Evolutionary Programming**
- Genetic Programming
- Coevolution
- Hybridized EA

# EP: historical perspective

- EP aimed at achieving intelligence.
- Artificial Intelligence Through Simulated Evolution.
- Intelligence viewed as adaptive behaviour.
- Prediction of the environment was considered a prerequisite to adaptive behaviour.
  - **Thus: capability to predict is key to intelligence**
- Differs substantially from GA and GP, in that EP emphasizes the development of behavioral linkage between Parents and their Offspring, rather than seeking to emulate specific genetic operators.
- EP considers phenotypic evolution.

# Motivation

Life on earth has evolved for some 3.5 billion years. Initially only the strongest creatures survived, but over time some creatures developed the ability to recall past series of events and apply that knowledge towards making intelligent decisions. The very existence of humans is testimony to the fact that our ancestors were able to outwit, rather than out power, those whom they were in competition with. This could be regarded as the beginning of intelligent behavior.

# Motivation (*contd*)


Simulated evolution is the process of duplicating certain aspects of the evolutionary system in the hopes that such an undertaking will produce artificially intelligent automata that are capable of solving problems in new and undiscovered ways → → → ***Prediction***.

# EP: quick overview

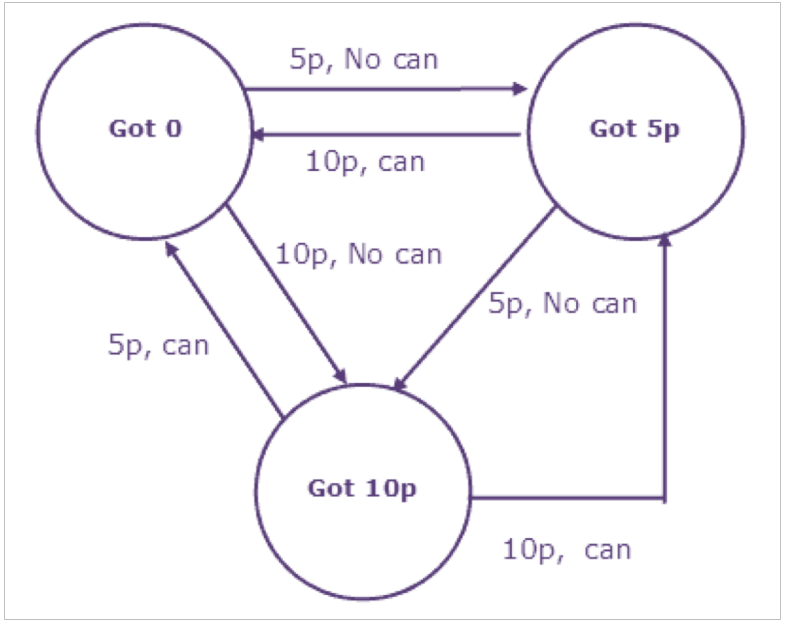
- Developed: USA in the 1960's
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: machine learning tasks by finite state machines.
  - contemporary EP: (numerical) optimization.
- Attributed features:
  - very open framework: any representation and mutation op's OK.
  - crossbred with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

# Prediction by Finite State Machines (FSM)

- Finite state machine (FSM):
  - States S
  - Inputs I
  - Outputs O
  - Transition function $\delta$ : S x I $\rightarrow$ S x O
  - Transforms input stream into output stream

- Can be used for predictions, e.g. to predict next input symbol in a sequence

# Vending Machine




Vending machine sells water for 15p

States:

1. Got 0
2. Got 5p
3. Got 10p

Inputs:

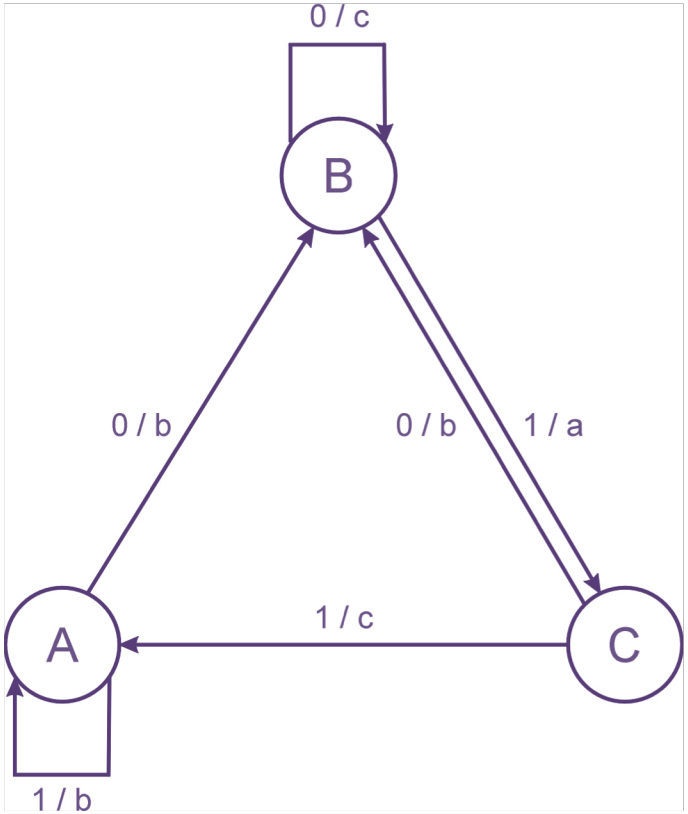
1. 5p
2. 10p

Output:

- can of water

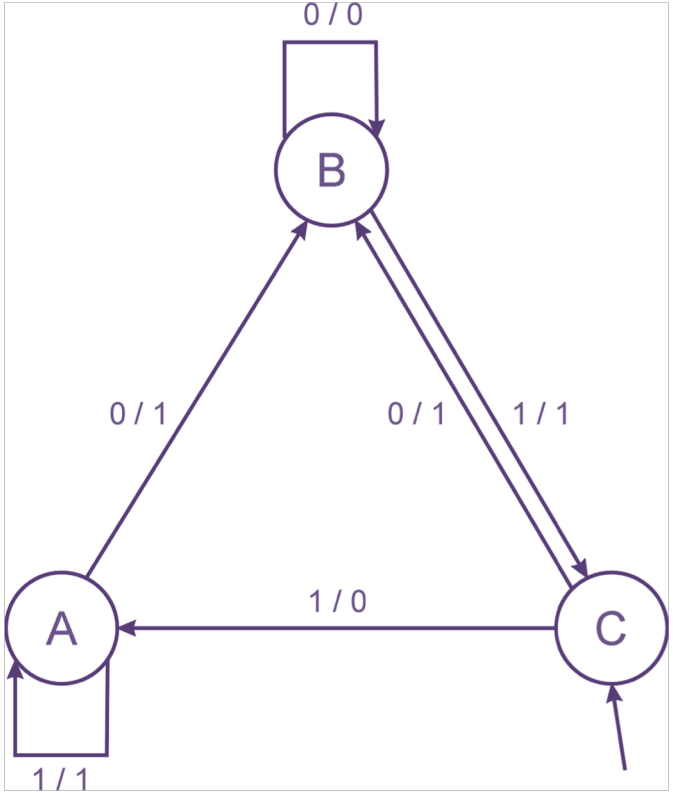
# FSM Example

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



# FSM as Predictor

- 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



# Evolving FSMs to Predict Primes (*Fogel et al.*)

- 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 much” states

# Evolving FSMs to Predict Primes (*Fogel et al.*)

- 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: $(\mu+\mu)$ : after having created $\mu$ offspring from a population of $\mu$ FSMs, the top 50% of their union is saved as the next generation.
- Results: overfitting, after 202 inputs best FSM had one state and both outputs were 0, i.e., it always predicted “not prime”

# Basic EP Algorithm

$t = 0$, initialize strategy parameters;

Create and initialize the population, $C(0)$, of $n_s$ individuals;

Evaluate the fitness, $f(x_i(t))$, of each individual;

**while** stopping condition(s) not true **do**

    **for** each individual, $x_i(t) \in C(t)$ **do**

        Create an offspring, $x_i'(t)$, by applying the mutation operator;

        Evaluate the fitness, $f(x_i'(t))$;

        Add $x_i'(t)$ to the set of offspring, $C'(t)$;

    **end**

    Select the new population, $C(t+1)$, from $C(t) \cup C'(t)$;

    $t = t + 1$;

**end**

# EP: Evaluation

- Survival in EP is usually based on a relative fitness measure.
- A score is computed to quantify how well an individual compares with a randomly selected group of competing individuals:
  - An individual is compared several times with a group of sub-population (created from the pool of both parents and off-spring).
  - A score is given to the individual based on his winning in the comparisons against the selected sub-population.
  - The same procedure is repeated for every individual.
  - Then, tournament selection based on the fitness described earlier.
- This procedure ensures "***better-ness***" in respect of the whole population.

# Basic Components

Traditional EP

- Mutation as the only source of variation.
- Selection
  - Main purpose to select new population.
  - Competitive process where parents and offspring compete to Survive.
- Behaviors of individuals are influenced by strategy parameters.

Modern EP:

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

# Representation

- For continuous parameter optimisation.
- Chromosomes consist of two parts:

$$\langle \underbrace{x_1, \ldots, x_n}_{\bar{x}}, \underbrace{\sigma_1, \ldots, \sigma_n}_{\bar{\sigma}} \rangle$$

  - Object variables: $x_1, \ldots, x_n$
  - Strategy Parameters/Mutation step sizes: $\sigma_1, \ldots, \sigma_n$

# Mutation

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n\rangle$ into $\langle x'_1,\ldots,x'_n, \sigma'_1,\ldots,\sigma'_n\rangle$.

  $\sigma_i' = \sigma_i \bullet (1 + \alpha \bullet N(0,1))$.

  $x'_i = x_i + \sigma_i' \bullet N_i(0,1)$.

  $\alpha \approx 0.2$.

  boundary rule: $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

- Other variants proposed & tried:
  - Lognormal scheme as in ES.
  - Using variance instead of standard deviation.
  - Mutate $\sigma$-last.
  - Other distributions, e.g, Cauchy instead of Gaussian

# Strategy Parameters

- The deviations, $\sigma_{ij}$ , are referred to as strategy parameters.
- Each individual has its own strategy parameters.
- An individual is represented as the tuple,

$$x_i(t) = (x_i(t), \sigma_I(t))$$

- Correlation coefficients between components of the individual have also been used as strategy parameters

# 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” Pragmatic approach seems to prevail today.

# Parent Selection

- *Almost nonissue.*
  - This distinguishes EP from other EA dialects.
- Each individual creates one child by mutation.
- Thus:
  - Deterministic.
  - Not biased by fitness.

# Survivor Selection: discussed earlier

- The selection operator is generally ($\mu$ + $\mu$) selection.
- *P(t):* $\mu$ parents, *P'(t):* $\mu$ offspring.
- Pairwise competitions in round-robin format (between one individual and *q* other randomly chosen solutions) :
  - Each solution *x* from $P(t) \cup P'(t)$ is evaluated against *q* other randomly chosen solutions.
  - For each comparison, a "win" is assigned if *x* is better than its opponent.
- Parameter *q* allows tuning selection pressure (Typically $q = 10$).
- The $\mu$ solutions with the greatest number of wins are retained to be parents of the next generation (described later).

# EP Technical Summary Tableau

| | |
|---|---|
| Representation | Real-valued vectors |
| Recombination | None |
| Mutation | Gaussian perturbation |
| Parent selection | Deterministic |
| Survivor selection | Probabilistic ($\mu$+$\mu$) |
| Specialty | Self-adaptation of mutation step sizes (in meta-EP) |

# EP and GA

Representation:

- GA requires the solution attempt to be encoded in a string of values.
- EP uses whatever representation fits the problem.

Mutation:

- In EP, it is a normally distributed perturbation.
    - Has infrequent large changes, frequent small changes.
    - Mutation rate decays as run time elapses
- GA mutation tends to be fixed size changes that create entirely new values

NTNU

# EP and ES

- EP and ES are very similar.
- Major differences are selection and recombination.
- Selection in EP is stochastic
  - tournament based
  - randomly selected participants
- Selection in ES is deterministic
  - bad individuals are purged
  - good individuals breed
  - no randomness involved
- No recombination is used in EP, Multi individual ES will use recombination

NTNU

# Outline

- Evolution Strategies
- Evolutionary Programming
- **Genetic Programming**
- Coevolution
- Hybridized EA

# Motivation

- One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it.
- GP addresses this challenge by providing a method for automatically creating a working computer program from a high-level problem statement of the problem.
- GP achieves this goal of automatic programming by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations.

Collected from John R. Koza

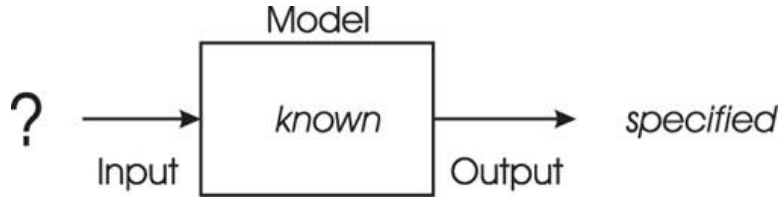
# Genetic Programming (GP)

- GP is a technique whereby computer programs are encoded as a set of genes that are then modified (evolved) using an EA.
- it is an application of (for example) GA where the space of solutions consists of computer programs.
- The results are computer programs able to perform well in a predefined task.
- **Does not make distinction between search and solution space.**
- Solution represented in very specific hierarchical manner.

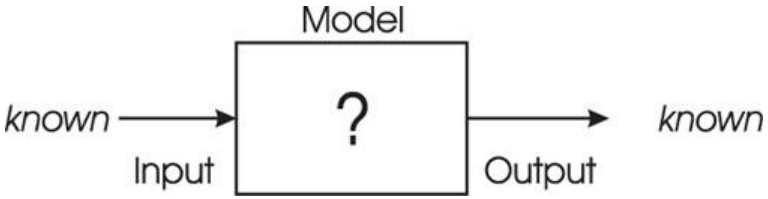
# GP Quick Overview

- Developed: USA in the 1990's.
- Early names: J. Koza.
- Typically applied to:
  - machine learning tasks (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 (disputed!) .

# GP in Machine Learning




Most Other EAs are for finding some input realising maximum payoff.




GP seeks models with maximum fit.

- **EAs discussed so far are typically applied to optimization problems**,
  - GP could instead be positioned **in machine learning**.

# Example: Credit Scoring

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

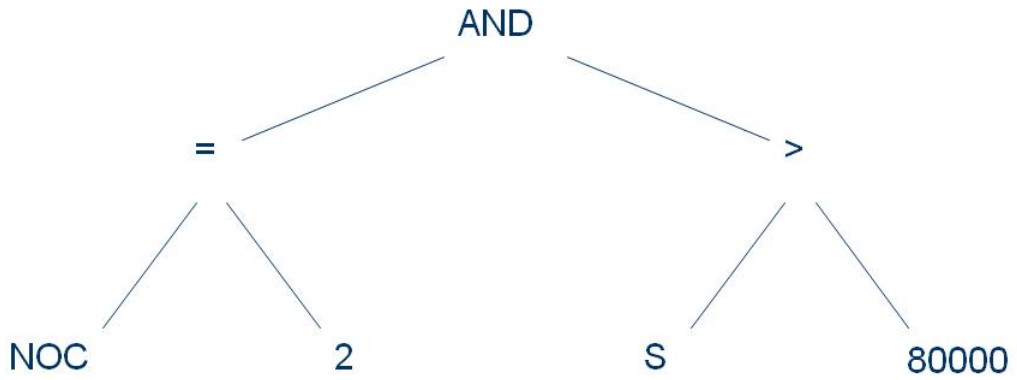
| ID | No of children | Salary | Marital status | OK? |
|---|---|---|---|---|
| ID-1 | 2 | 45000 | Married | 0 |
| ID-2 | 0 | 30000 | Single | 1 |
| ID-3 | 1 | 40000 | Divorced | 1 |
| … | | | | |

# Example: Credit Scoring

- 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
  - Search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees.

# Example: Credit Scoring

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



# Abstract GP algorithm

1: Randomly create an *initial population* of programs from available primitives.

2: **repeat**

3: *Execute* each program and ascertain its fitness.

4: *Select* one or two program(s) from the population with a probability based on fitness to participate in genetic operations.

5: Create new individual program(s) by applying *genetic operations* with specified probabilities.

6: **until** an acceptable solution is found or some other stopping condition is met (for example, reaching a maximum number of generations).

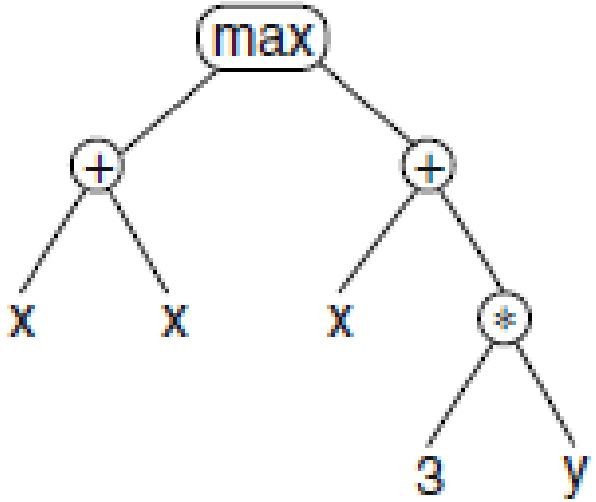
7: **return** the best-so-far individual

# Tree Structure

- Terminal Nodes: the nodes that make up the leaves of a program tree, provide data to the program.
  - Constants.
  - Parameterless functions.
  - Inputs
- Non Terminal nodes: Function set
  - Set of available functions.
  - Often tailored specifically for the needs of the program domain.

# Tree Structure of GP

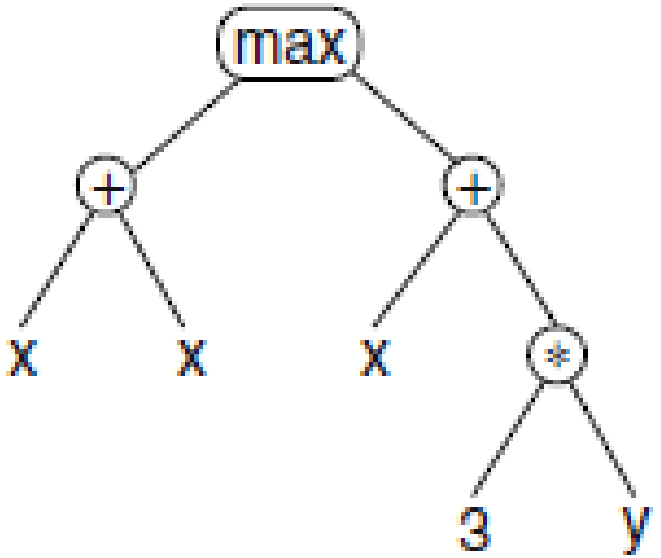
- In GP programs are usually expressed as syntax trees rather than as lines of code.
- Figure shows the tree representation of the program **max**$(x+x,x+3*y)$.
- Variables and constants in the program ($x$, $y$, and $3$), called terminals in GP, are leaves of the tree.
- Arithmetic operations (+, *, and max) are internal nodes (typically called functions).
- The sets of allowed functions and terminals together form the primitive set of a GP system.




GP syntax tree representing max(x+x,x+3y)

# Tree Structure of GP

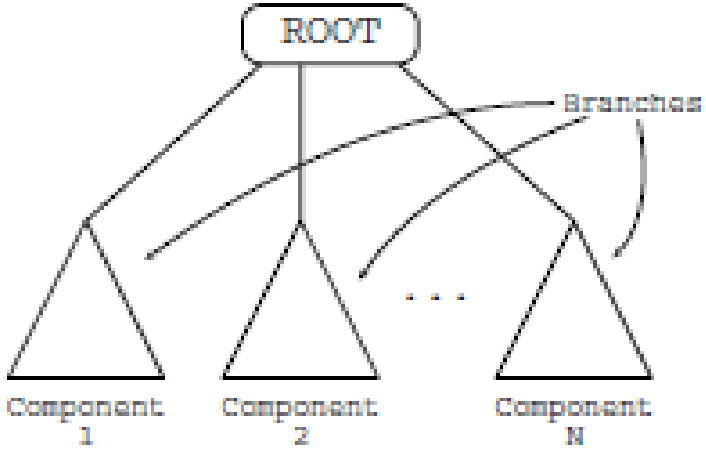
- It is common in the GP literature to represent expressions in a *prefix* notation similar to that used in Lisp or Scheme.
- For example, max(x+x,x+3*y) becomes (max (+ x x) (+ x (* 3 y))).




GP syntax tree representing max(x*x,x+3y)

# Tree Structure of GP

- In more advanced forms of GP, programs can be composed of multiple components (say, subroutines).
- In this case the representation used in GP is a set of trees (one for each component) grouped together under a special root node that acts as glue.




Multi-component program representation

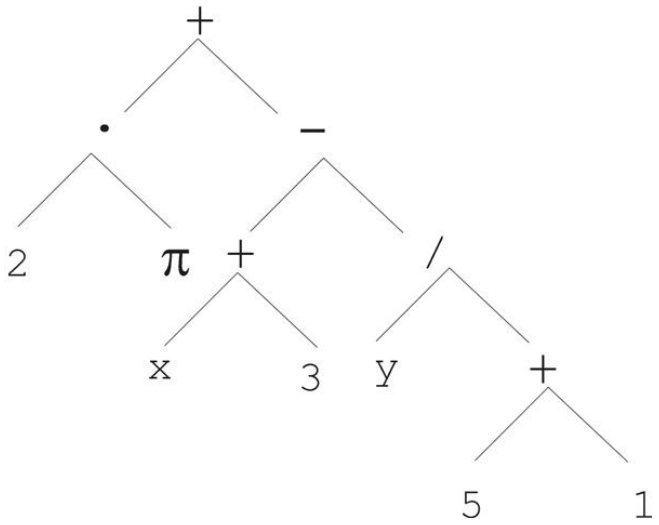
# Tree based Representation

Trees are a universal form, e.g. consider

- Arithmetic formula $2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$
- Logical formula $(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program

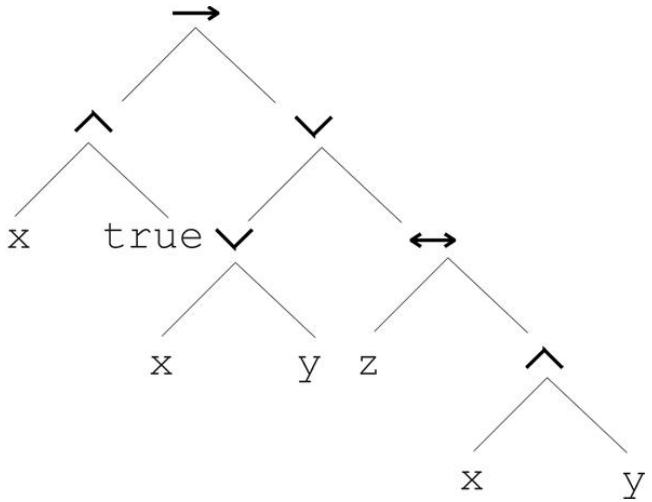
```
i = 1;
while (i < 20)
{
        i = i + 1
}
```

# Tree based Representation



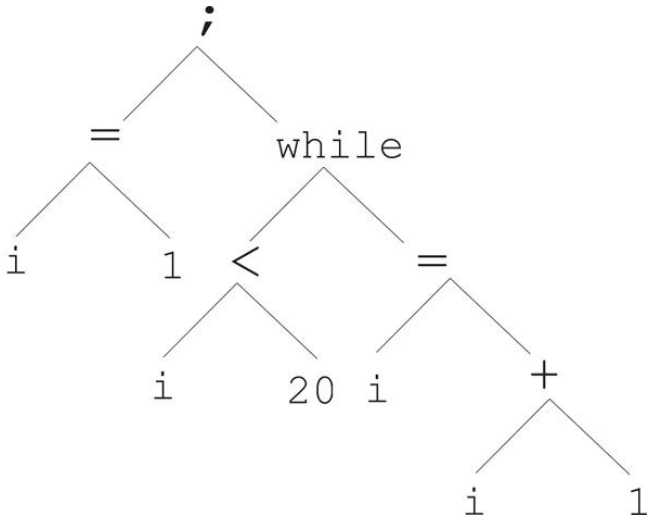

$$2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$$

# Tree based Representation



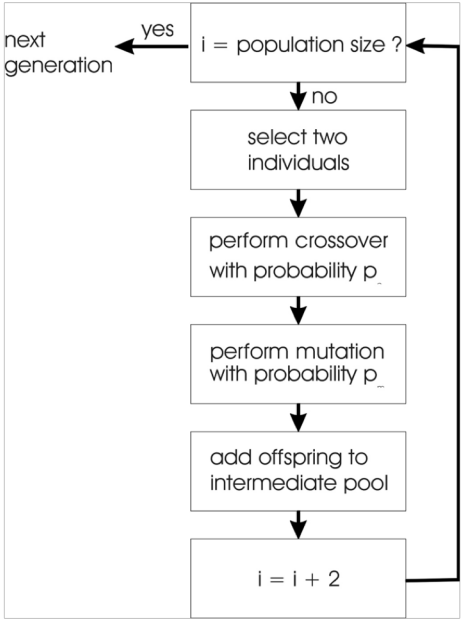

$(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

# Tree based Representation

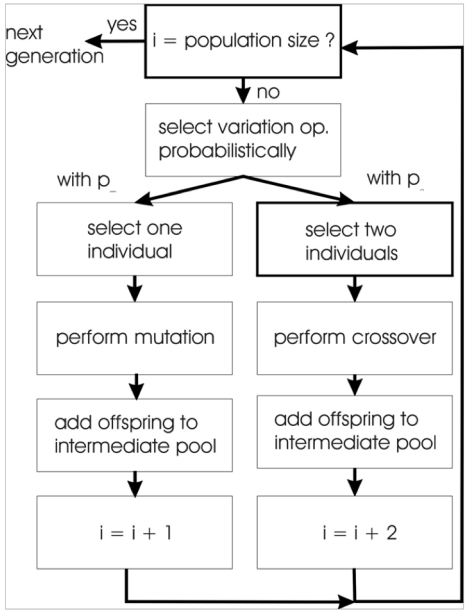



```
i = 1;
while (i < 20)
{
        i = i + 1
}
```

# Offspring Creation Scheme




GA flowchart




GP flowchart

- GA scheme use crossover AND mutation sequentially (be it probabilistically).
- GP scheme use either crossover OR mutation (chosen probabilistically).

# Initializing the Population

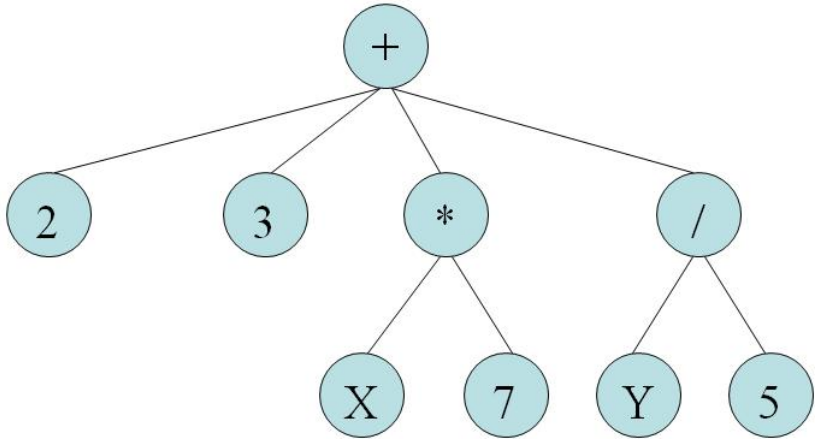
- The following two parameters are specified:
  - Maximum depth of a program tree.
  - Maximum number of nodes in a program tree.

# Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree.

# Mutation

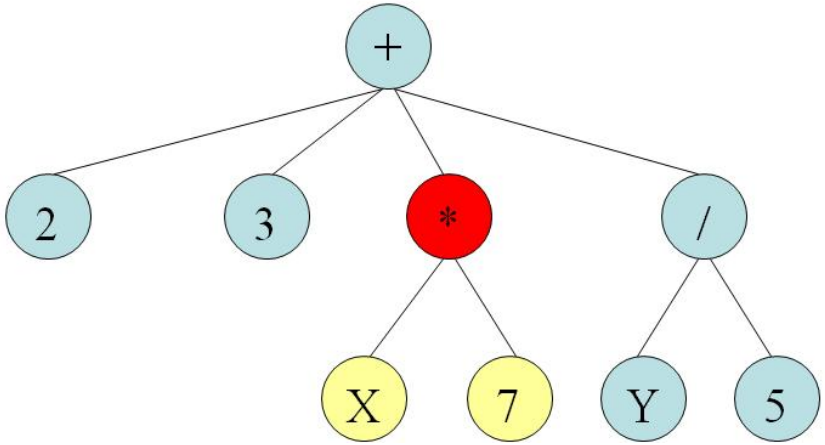
(+ 2 3 (* X 7) (/ Y 5))



# Mutation

(+ 2 3 (* X 7) (/ Y 5))

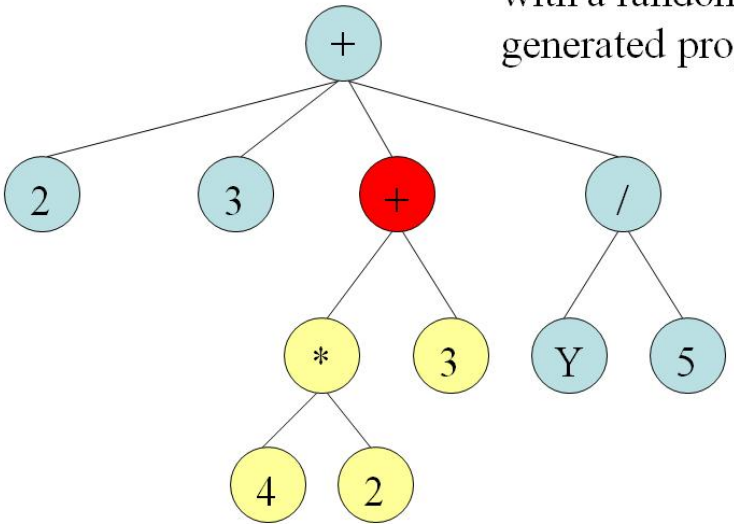
First pick a random node



# Mutation

(+ 2 3 (+ (* 4 2) 3) (/ Y 5))

Delete the node and its children, and replace with a randomly generated program



# Mutation

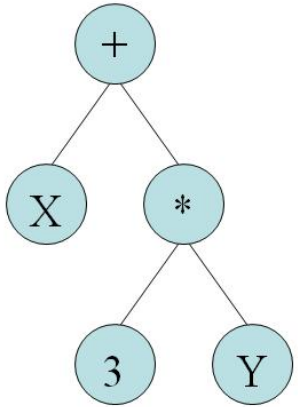
- Mutation has two parameters:
  - Probability $p_m$ to choose mutation vs. recombination.
  - Probability to chose an internal point as the root of the subtree to be replace.
- Remarkably $p_m$ is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98).
- The size of the child can exceed the size of the parent.

# Recombination

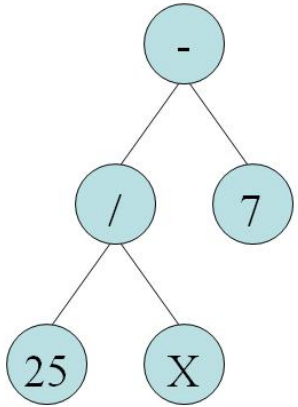
- Most common recombination: exchange two randomly chosen subtrees among the parents.
- Recombination has two parameters:
  - Probability $p_c$ to choose recombination vs. mutation.
  - Probability to chose an internal point within each parent as crossover point.
- The size of offspring can exceed that of the parents.

# Recombination

(+ X (* 3 Y))

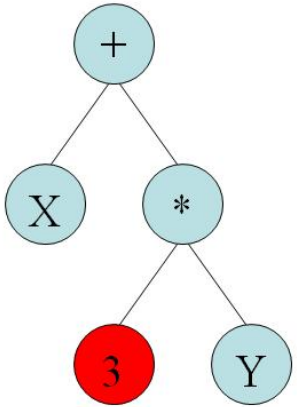



(- (/ 25 X) 7)



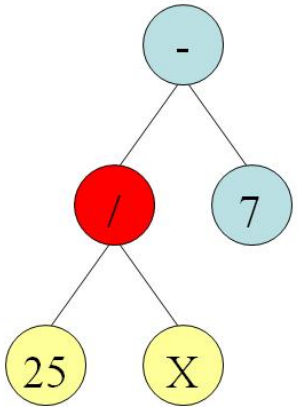
# Recombination

(+ X (* 3 Y))



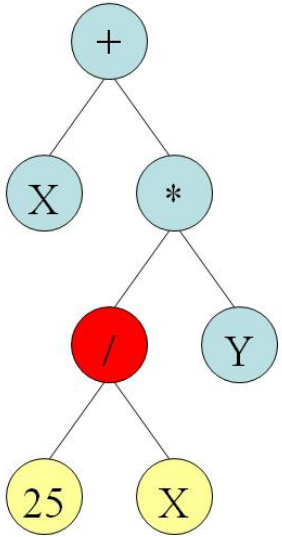

Pick a random node in each program

(- (/ 25 X) 7)



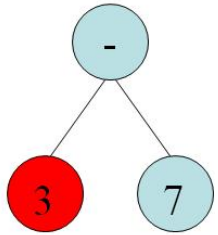
# Recombination

(+ X (* (/ 25 X) Y))




Swap the two nodes

(- 3 7)



# Parent 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).
  - No elitism.
  - The number of offspring created is the same as the population size.
  - All individuals have a life span of one generation.

- Recently steady-state is becoming popular for its elitism.

# Bloat

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

# GP Technical Summary Tableau

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

NTNU

# Real World Applications

- Neural Network Optimization.
- Image Analysis.
- Generation of a knowledge base for expert systems.
- Fuzzy Logic Control.
- Hardware Evolution (Field-Programmable Gate Array).

NTNU

# GA and GP

- Mechanistically similar to GA
- One major difference
  - no genotype / phenotype distinction
  - evolutionary operations carried out directly on the candidate programs themselves

# GP and Other EAs

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations).
- Tree shaped chromosomes are non-linear structures.

- In GA, ES, EP the size of the chromosomes is fixed.
- Trees in GP may vary in depth and width.

NTNU

# Outline

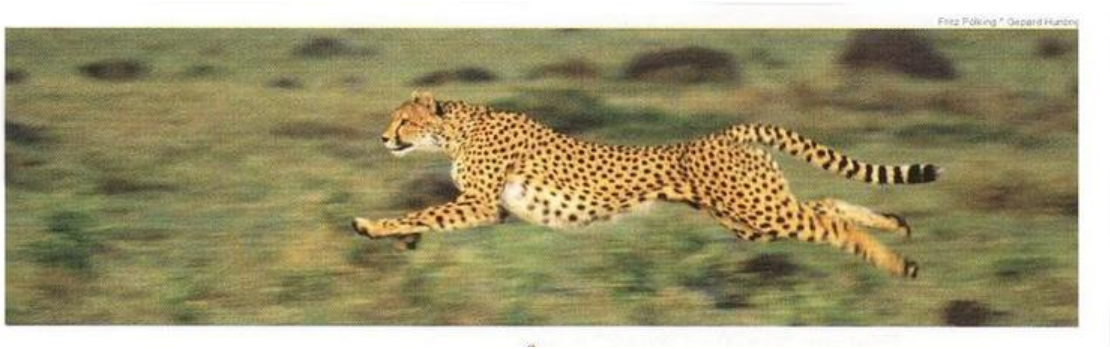
- Evolution Strategies
- Evolutionary Programming
- Genetic Programming
- **Coevolution**
- Hybridized EA

NTNU

# Fundamental Question …….

How could we possibly build an effective algorithm when any time we believe A is better than B --- it is possible that later we will believe B is better than A?

F.Polking (http://www.art.com/asp/sp.asp?PD=10056007#)

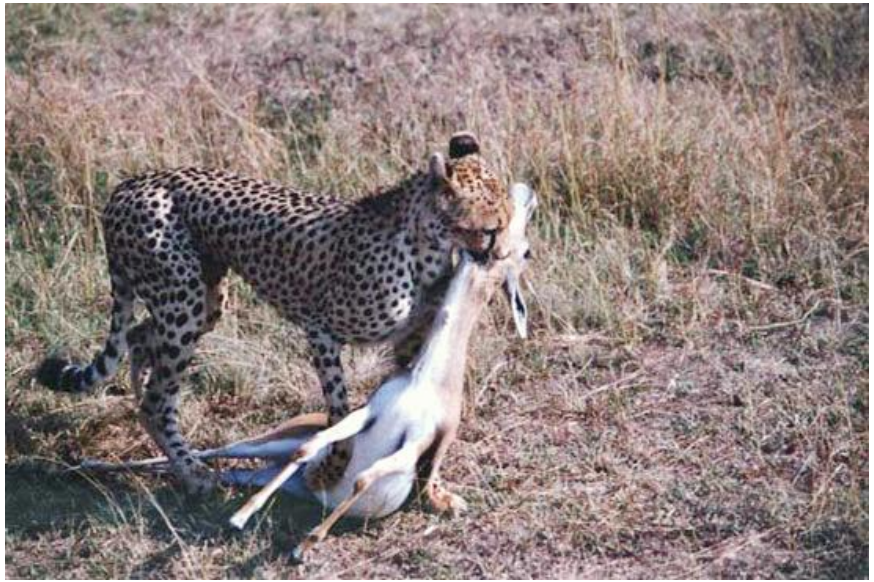


Greg Koch (www.nczoo.org/Photo_Gallery/GregKoch/Tommy2GregKoch_JPG.htm)

Cheetah: 60-70 mph, for up to 100 yards

highest recorded speed, 71 mph, can be sustained for about 300 yards.

can run more than 3 miles -- at an average speed of 45 mph.



Thompson's Gazelle: 50 mph...

Robb Hoenick (www.inkburns.com/html/cheetah.html)

The outcome (but about 50% of attempts are failures)

# Coevolutionary Algorithm (CoEA)

- CoEAs explore domains in which no single evaluation function is present or known.
- Instead, algorithms rely on the aggregation of outcomes from interactions among coevolving entities to make selection decisions.
- A special type of EAs where the fitness of an individual is dependent on other individuals.
  - Fitness landscape changes
  - Fitness of an individual may be different in different runs
- Change in one individual will change the fitness landscape in others

NTNU

# Coevolutionary Algorithm (CoEA)

**Very similar to traditional EA methods:**

- Individuals encode aspect of potential solutions
- They are altered during search with genetic operators
- Search directed by selection based on fitness

**But differ in fundamental ways:**

- Evaluation requires interaction between multiple individuals
- Interacting individuals may reside in same population or in different populations
- Evokes notions of cooperation and competition in new ways
- Methods of evaluation are particularly important

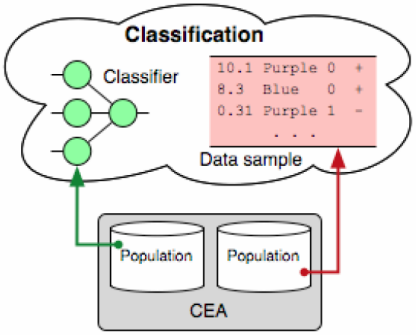
# CoEA: evaluation

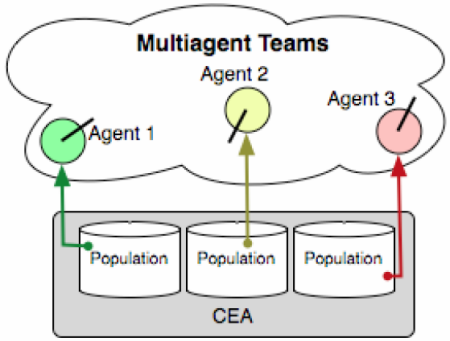
Form of evolutionary computation in which the fitness evaluation is based on interactions between multiple individuals

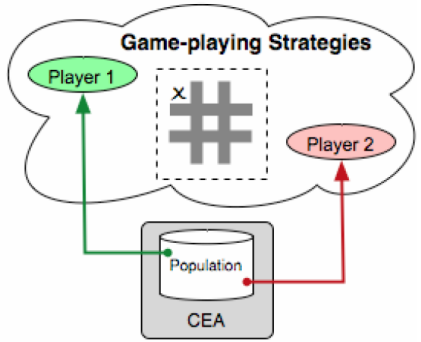
- Changes in the set of individuals used for evaluation can affect the ranking of individuals.
- Coevolutionary fitness is *absolute* or *subjective*.
- Traditional EC fitness is ***relative*** or ***objective***.

# Why Co-Evolution?

- No fitness-function (objective measures) known.
- Too many fitness cases.
- Certain types of structure in search space.
  - Modularizable Problem.
- Large (infinite) search spaces.



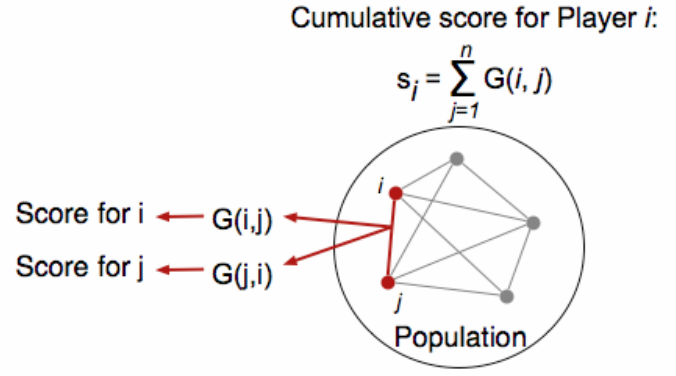
# Types of co-evolution

- By evaluation
  - Competitive
  - Cooperative

- By population-organisation
  - Single species
  - Multiple species

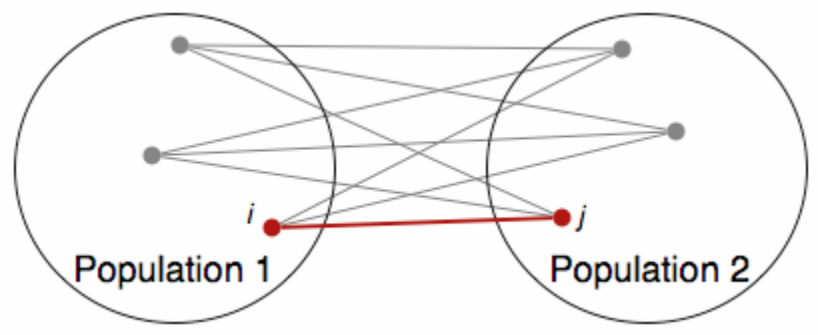
# Single Species (population) CoEA

- Individuals are evaluated by interacting with other individuals from that population.
- Employs a single EA, typically using traditional operators.
- Individuals represent candidate solutions to a problem, as well as “tests” for other individuals.
- Individuals compete in game-theoretic sense, but also compete as resources for evolution.



# Multiple Species (population) CoEA

- Individuals in one population interact with individuals in one or several other populations.
- Multiple EAs, one for each population.
- Individuals may represent a variety of things.
- Typically individuals in different populations do not compete directly for the EA resources



# Competetive/Cooperative CoEA

- In CoEA, individuals are evaluated based on their interactions with other individuals. This classification is according to the nature of these interactions.

**Cooperative algorithms:**

- Individuals are rewarded when they work well with other individuals and punished when they perform poorly together.
- Interacting individuals succeed or fail together

**Competitive algorithms:**

- Individuals are rewarded at the expense of those with which they interact.

NTNU

# Summary

- Co-Evolution is Everywhere.
- Can be cooperative and competitive.
- Can be in one population, or more than one.
- An individual's fitness is not fixed in co-evolution.
- Co-evolution is not well explored in Evolutionary Computation.

NTNU

# Outline

- Evolution Strategies
- Evolutionary Programming
- Genetic Programming
- Coevolution
- **Hybridized EA**

NTNU

# Why Hybridise

- Might want to put in EA as part of larger system.
- Might be looking to improve on existing techniques but not re-invent wheel.
- Might be looking to improve EA search for good solutions.

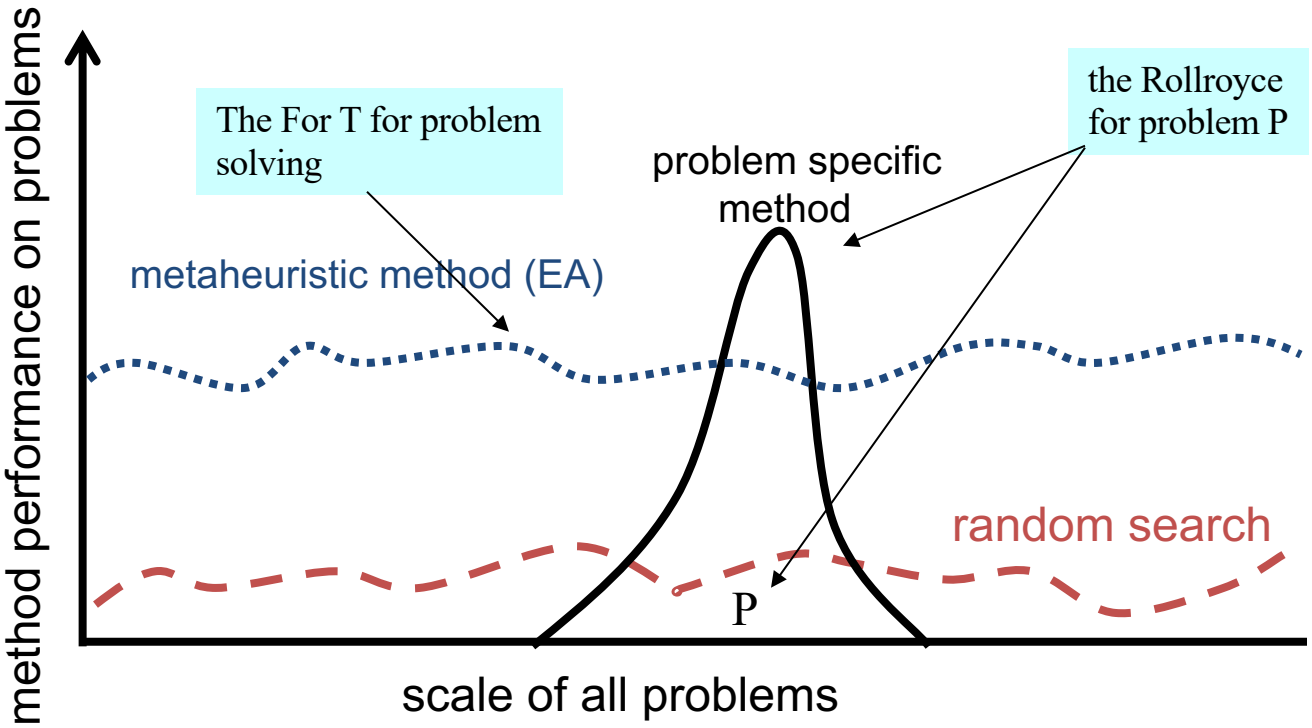
NTNU

# Why Hybridise

- Complex problems can be decomposed into a number of parts,
  - For some of which exact methods, or very good heuristics, may already be available.

- No overall successful and efficient general problem solver.
  - No Free Lunch (NFL) theorem supports this view.

# EAs as Problem Solvers: The Pre-90s View



# EAs as Problem Solvers: The Pre-90s View

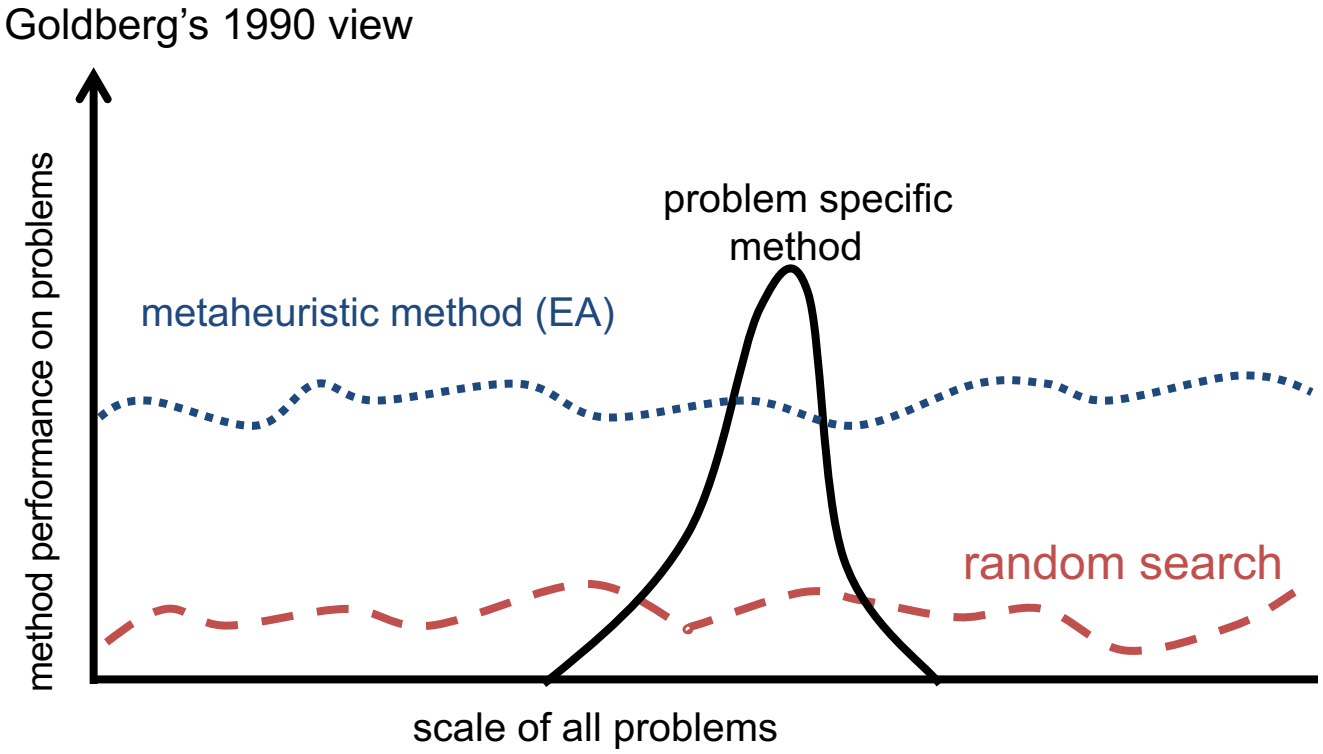
- EAs and problem-specific algorithms form two opposing extremes.

# EAs and Domain Knowledge

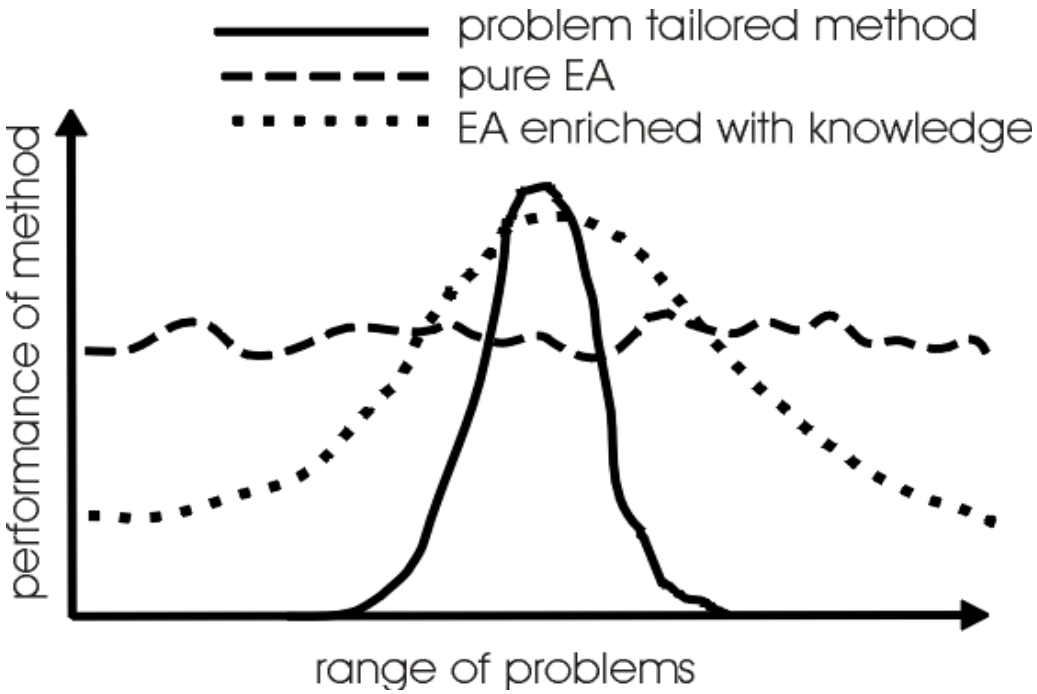
- Fashionable after the 90's:
  to add problem specific information into the EAs by means of specialized crossover, mutation, representations and local search.

- Result: The performance curve deforms and
  - makes EAs better in some problems,
  - worst on other problems
  - amount of problem specific is varied.

# The 90s View (Michalewicz's) on EAs in Context



# The 90s View (Michalewicz's) on EAs in Context



# The 90s View (Michalewicz's) on EAs in Context

- The figures considers the possibility to combine problem specific heuristics and an EA into a hybrid algorithm.
- The amount of problem-specific knowledge is variable and can be adjusted.
- The global performance curve will gradually change from roughly flat (pure EA) to a narrow peak (problem-specific method).

# Required Reading + References

- Eiben: chapters: 10 (10.2); 4 (4.4, 4.8, 4.9); 5 (5.6); 6 (6.11)

- References:
  - [Fogel et al.] L.J. Fogel, A.J. Owens, M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK, 1966