Introduction to Evolutionary Computation

Marc Segond

Presentation

- Marc Segond, 34 years old, French
- PhD in Computer Science in 2006 (ULCO)
- Since end of 2012 Post-doctoral researcher / Research and Development engineer at Ambrosys GmbH (Germany)
- 4 years post-doctoral researcher at the European Center for Soft Computing (Spain) 2008 - 2012
- Expertise: Bio-mimetic Algorithms, Intelligent Data Mining, Graph Mining

ambrosys

Contents

- What is Evolutionary Computation?
  - History
  - biological principles
  - application to computer science
- Genetic Algorithms
  - Details on mechanisms
  - Few toy examples
- Genetic Programming
  - Details on mechanisms
  - Real life problem example

Positioning of EC

- EC is part of computer science
- EC is not part of life sciences/biology
- Biology delivered inspiration and terminology
- EC can be applied in biological research

The Main Evolutionary Computing Metaphor

EVOLUTION		PROBLEM SOLVING
Environment	⟷	Problem
Individual	⟷	Candidate Solution
Fitness	⟷	Quality

Fitness → chances for survival and reproduction

Quality → chance for seeding new solutions

Brief History 1: the ancestors

- 1948, Turing:
  proposes "genetical or evolutionary search"
- 1962, Bremermann
  optimization through evolution and recombination
- 1964, Rechenberg
  introduces evolution strategies
- 1965, L. Fogel, Owens and Walsh
  introduce evolutionary programming
- 1975, Holland
  introduces genetic algorithms
- 1992, Koza
  introduces genetic programming

Brief History 2: The rise of EC

- 1985: first international conference (ICGA)
- 1990: first international conference in Europe (PPSN)
- 1993: first scientific EC journal (MIT Press)
- 1997: launch of European EC Research Network EvoNet

EC in the early 21st Century

- 3 major EC conferences, about 10 small related ones
- 3 scientific core EC journals
- 750-1000 papers published in 2003 (estimate)
- EvoNet has over 150 member institutes
- uncountable (meaning: many) applications
- uncountable (meaning: ?) consultancy and R&D firms

Darwinian Evolution 1: Survival of the fittest

- All environments have finite resources

  (i.e., can only support a limited number of individuals)
- Lifeforms have basic instinct/ life-cycles geared towards reproduction
- Therefore some kind of selection is inevitable
- Those individuals that compete for the resources most effectively have increased chance of reproduction

Darwinian Evolution 2: Diversity drives change

- Phenotypic traits:
  - Behavior / physical differences that affect response to environment
  - Partly determined by inheritance, partly by factors during development
  - Unique to each individual, partly as a result of random changes
- If phenotypic traits:
  - Lead to higher chances of reproduction
  - Can be inherited

  then they will tend to increase in subsequent generations,
- leading to new combinations of traits …

Darwinian Evolution:Summary

- Population consists of diverse set of individuals
- Combinations of traits that are better adapted tend to increase representation in population

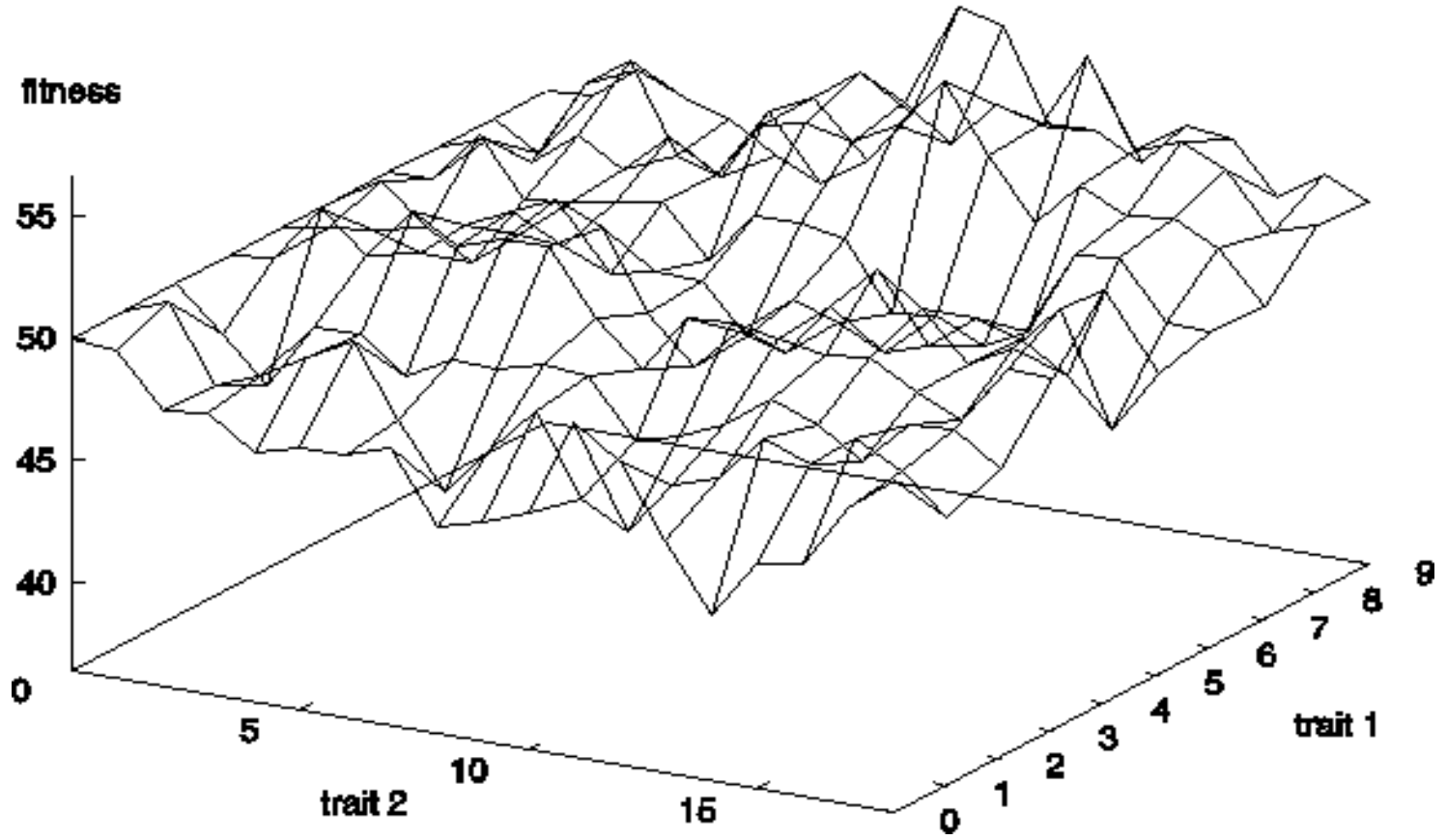
  Individuals are “units of selection”
- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:

  Population is the “unit of evolution”
- Note the absence of “guiding force”

Adaptive landscape metaphor (Wright, 1932)

- Can envisage population with *n* traits as existing in a *n+1*-dimensional space (landscape) with height corresponding to fitness
- Each different individual (phenotype) represents a single point on the landscape
- Population is therefore a “cloud” of points, moving on the landscape over time as it evolves - adaptation

Example with two traits



ambrosys

Adaptive landscape metaphor (cont'd)

- Selection "pushes" population up the landscape

- Genetic drift:
  - random variations in feature distribution
    (+ or -) arising from sampling error
  - can cause the population "melt down" hills, thus crossing valleys and leaving local optima

Natural Genetics

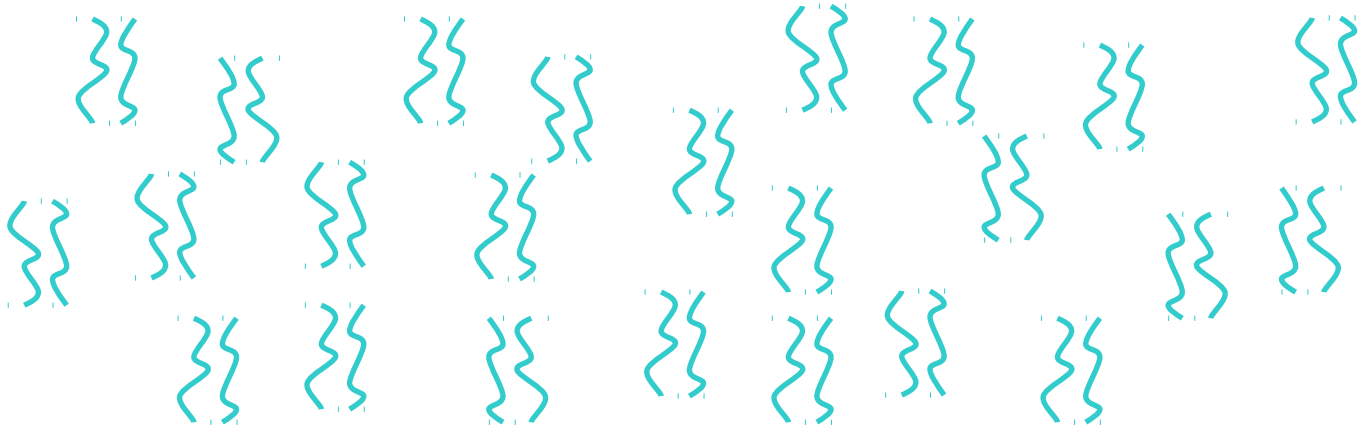
- The information required to build a living organism is coded in the DNA of that organism
- Genotype (DNA inside) determines phenotype
- Genes → phenotypic traits is a complex mapping
  - One gene may affect many traits (pleiotropy)
  - Many genes may affect one trait (polygeny)
- Small changes in the genotype lead to small changes in the organism (e.g., height, hair colour)

Genes and the Genome

- Genes are encoded in strands of DNA called chromosomes
- In most cells, there are two copies of each chromosome (diploidy)
- The complete genetic material in an individual's genotype is called the Genome
- Within a species, most of the genetic material is the same

Example: Homo Sapiens

- Human DNA is organized into chromosomes
- Human body cells contains 23 pairs of chromosomes which together define the physical attributes of the individual:

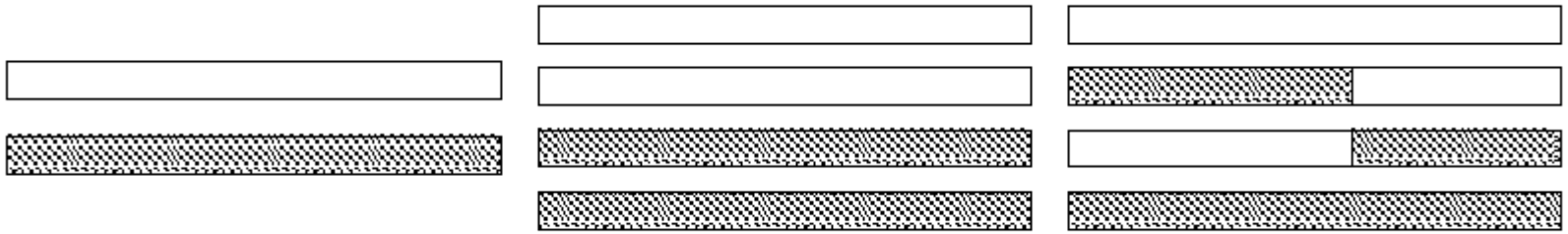


Reproductive Cells

- Gametes (sperm and egg cells) contain 23 individual chromosomes rather than 23 pairs
- Cells with only one copy of each chromosome are called Haploid
- Gametes are formed by a special form of cell splitting called meiosis
- During meiosis the pairs of chromosome undergo an operation called *crossing-over*

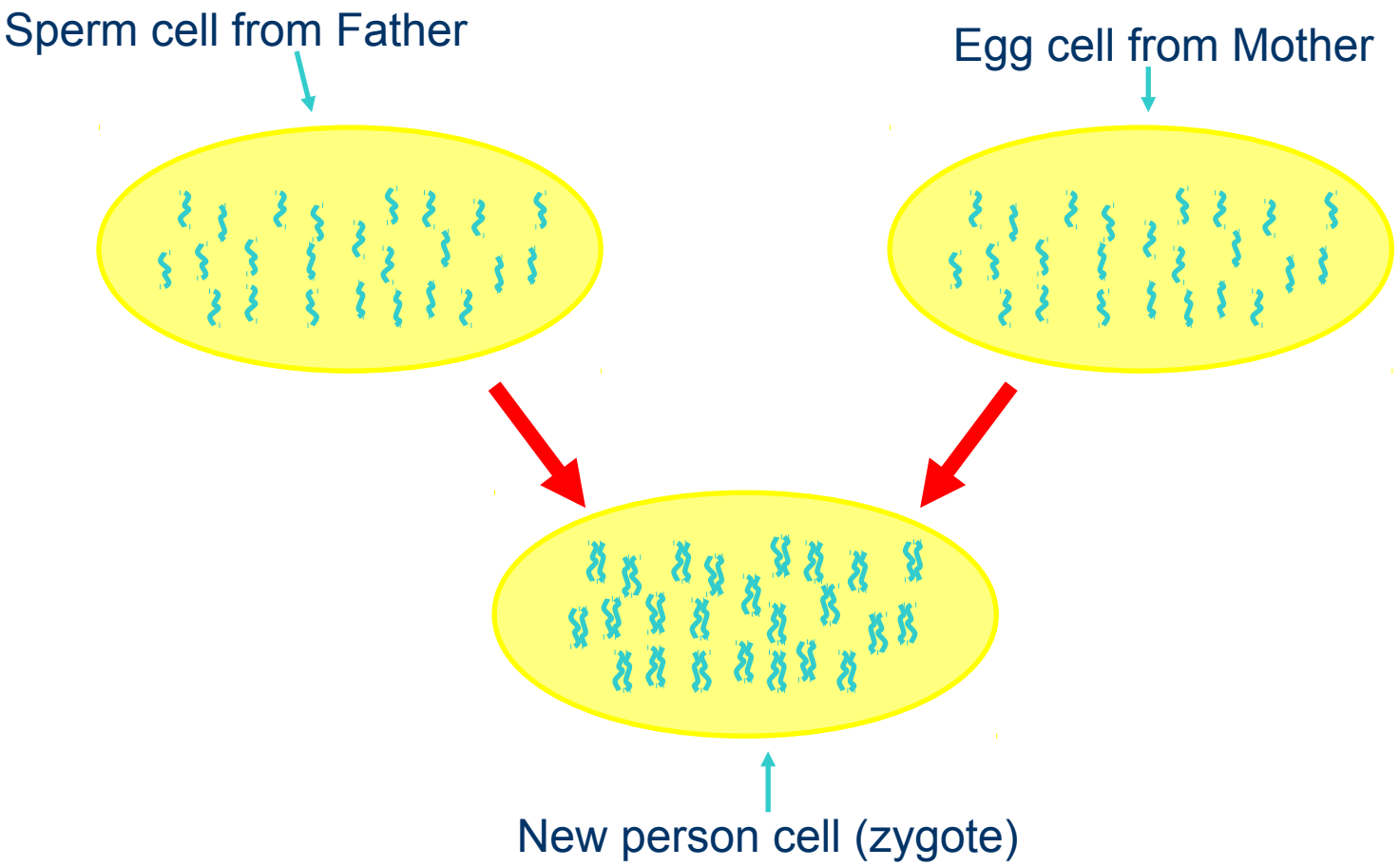
Crossing-over during meiosis

- Chromosome pairs align and duplicate
- Inner pairs link at a *centromere* and swap parts of themselves



- Outcome is one copy of maternal/paternal chromosome plus two entirely new combinations
- After crossing-over one of each pair goes into each gamete

Fertilisation



ambrosys

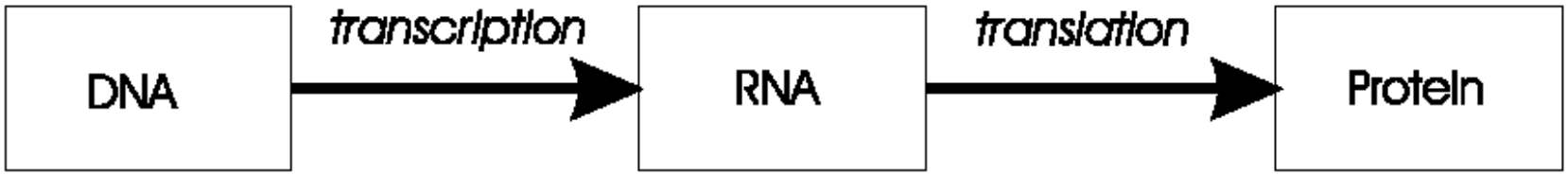
After fertilisation

- New zygote rapidly divides etc creating many cells all with the same genetic contents
- Although all cells contain the same genes, depending on, for example where they are in the organism, they will behave differently
- This process of differential behavior during development is called ontogenesis
- All of this uses, and is controlled by, the same mechanism for decoding the genes in DNA

Genetic code

- All proteins in life on earth are composed of sequences built from 20 different amino acids
- DNA is built from four nucleotides in a double helix spiral: purines A,G; pyrimidines T,C
- Triplets of these form *codons*, each of which codes for a specific amino acid
- Much redundancy:
  - purines complement pyrimidines
  - the DNA contains much rubbish
  - 4^3=64 codons code for 20 amino acids
  - genetic code = the mapping from codons to amino acids
- **For all natural life on earth, the genetic code is the same !**

Transcription, translation



A central claim in molecular genetics: only one way flow

Genotype → Phenotype

Genotype ↛ Phenotype

Lamarckism (saying that acquired features can be inherited) is thus wrong!

Mutation

- Occasionally some of the genetic material changes very slightly during this process (replication error)
- This means that the child might have genetic material information not inherited from either parent
- This can be
  - catastrophic: offspring in not viable (most likely)
  - neutral: new feature not influences fitness
  - advantageous: strong new feature occurs
- Redundancy in the genetic code forms a good way of error checking

Motivations for EC: 1

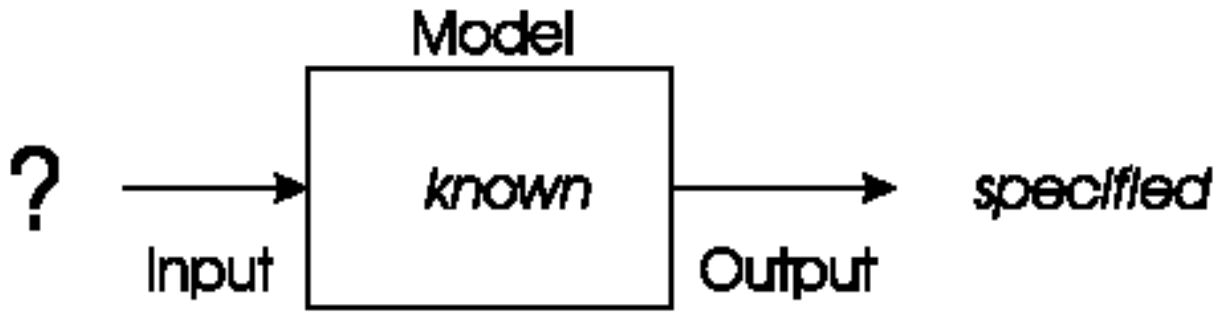
- Nature has always served as a source of inspiration for engineers and scientists
- The best problem solver known in nature is:
  - the (human) brain that created “the wheel, New York, wars and so on” (after Douglas Adams’ Hitch-Hikers Guide)
  - the evolution mechanism that created the human brain (after Darwin’s Origin of Species)
- Answer 1 → neurocomputing
- Answer 2 → evolutionary computing

Motivations for EC: 2

- Developing, analyzing, applying problem solving methods a.k.a. algorithms is a central theme in mathematics and computer science
- Time for thorough problem analysis decreases
- Complexity of problems to be solved increases
- Consequence:
  Robust problem solving technology needed

Problem type 1 : Optimization

- We have a model of our system and seek inputs that give us a specified goal



- e.g.
  - time tables for university, call center, or hospital
  - design specifications, etc etc

Optimization example: Satellite structure

Optimised satellite designs for NASA to maximize vibration isolation

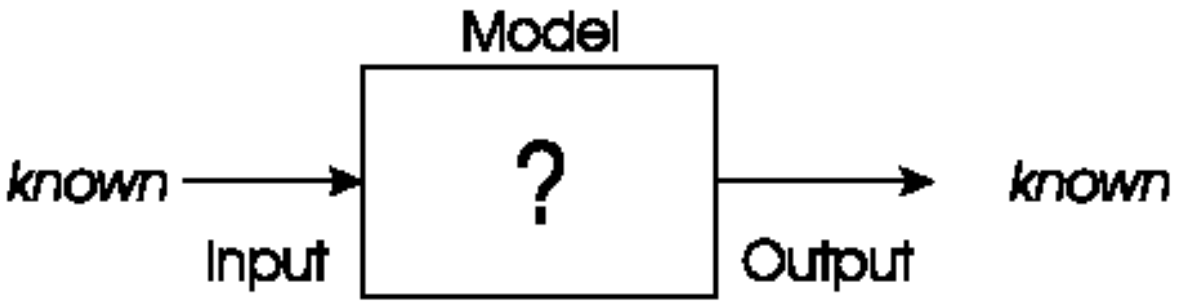
Evolving: design structures

Fitness: vibration resistance

Evolutionary “creativity”

Problem types 2: Modelling

- We have corresponding sets of inputs & outputs and seek model that delivers correct output for every known input



- Evolutionary machine learning

Modelling example: loan applicant creditibility

British bank evolved cred-itability model to predict loan paying behavior of new applicants

Evolving: prediction models

Fitness: model accuracy on historical data

Genetic Algorithms

GA Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

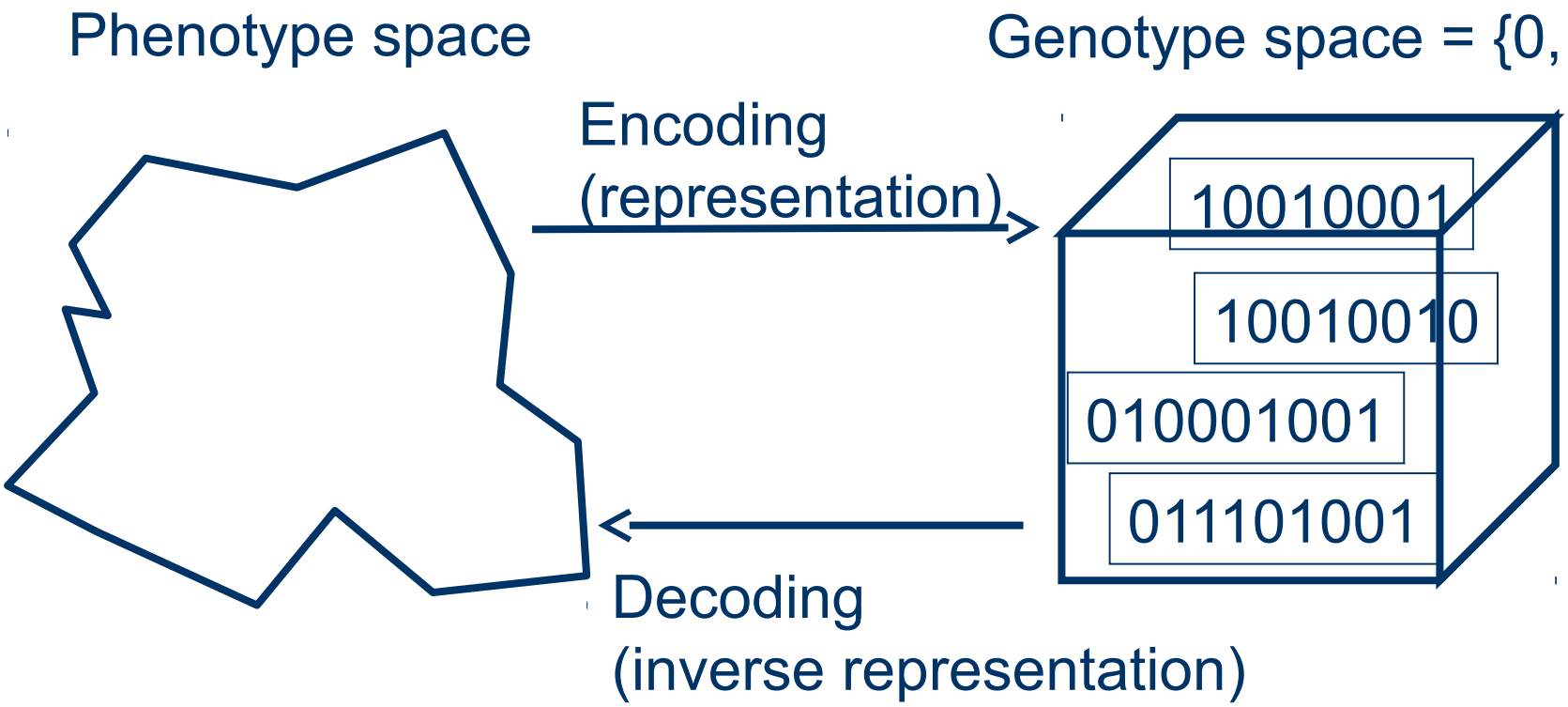
Genetic algorithms

- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Representation

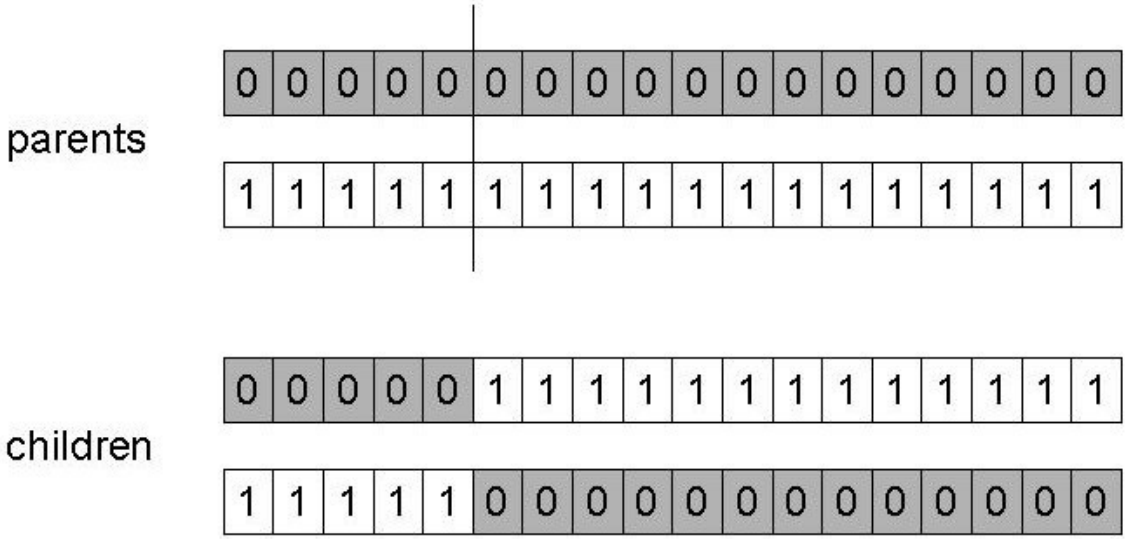


SGA reproduction cycle

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

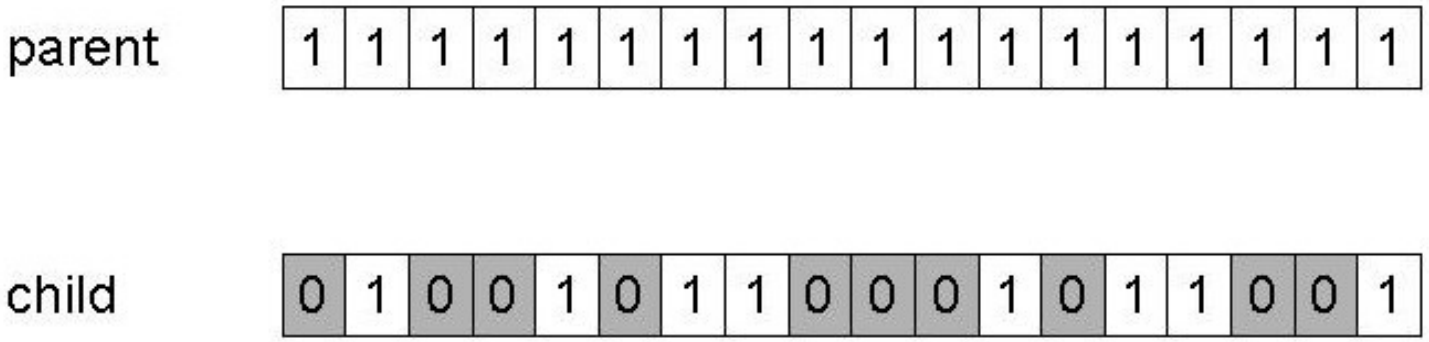
SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



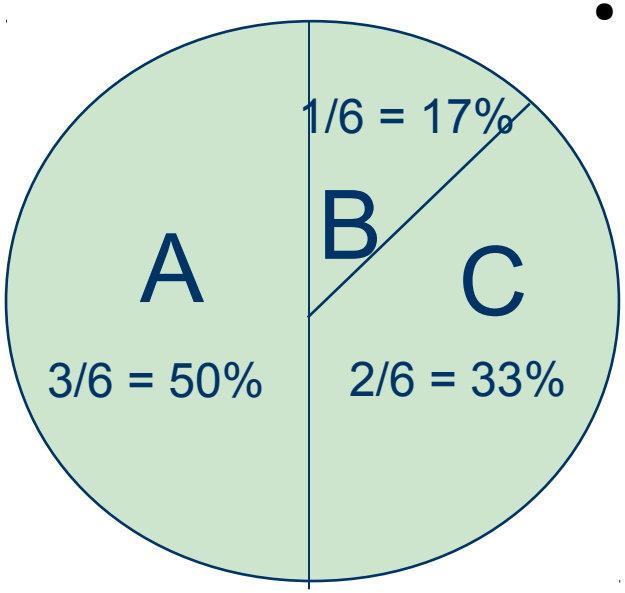
SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length



SGA operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - Assign to each individual a part of the roulette wheel
    - Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

An example after Goldberg ‘89 (1)

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show 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

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

The simple GA

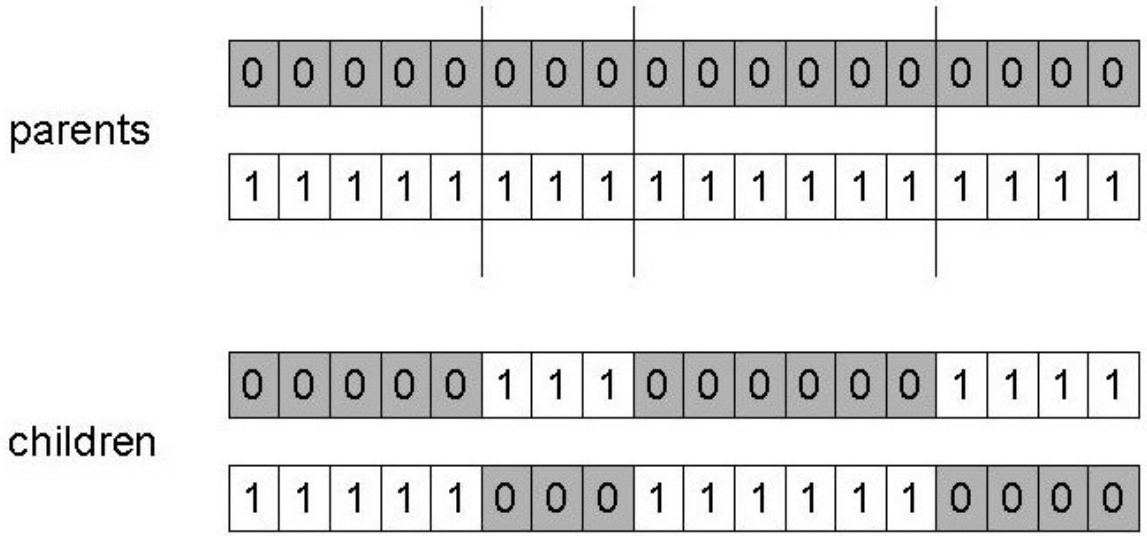
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

Alternative Crossover Operators

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
  - more likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string
  - This is known as *Positional Bias*
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

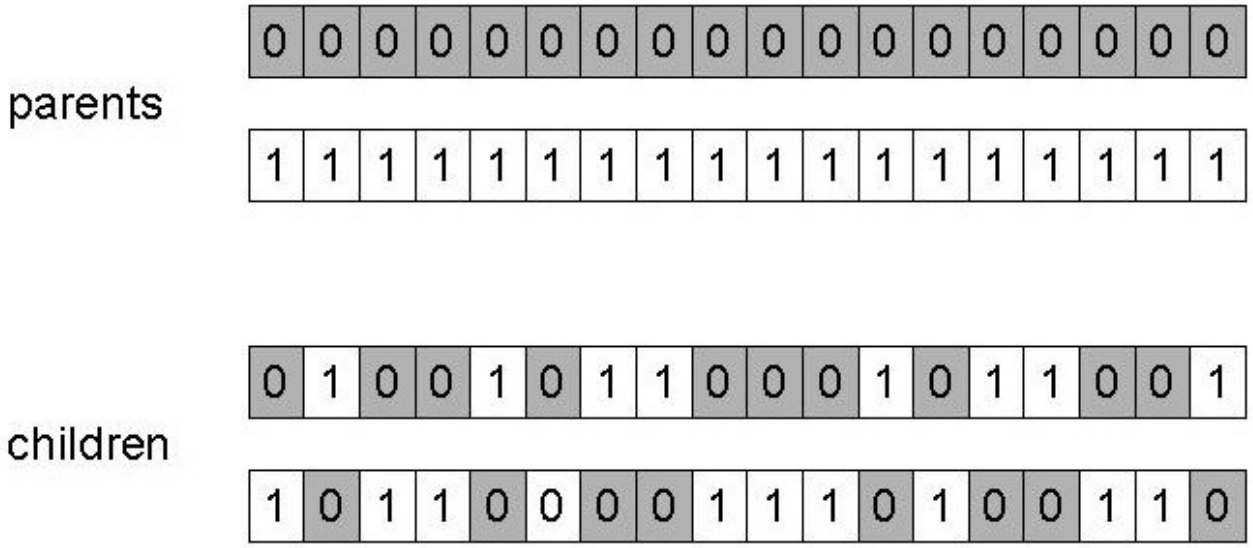
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)



Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



Crossover OR mutation?

- Decade long debate: which one is better / necessary / main-background

- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have a different role

Crossover OR mutation? (cont'd)

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

Crossover OR mutation? (cont'd)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, 50% after performing *n* crossovers)
- To hit the optimum you often need a 'lucky' mutation

Other representations

- Gray coding of integers (still binary chromosomes)
  - Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding). “Smoother” genotype-phenotype mapping makes life easier for the GA

Nowadays it is generally accepted that it is better to encode numerical variables directly as

- Integers
- Floating point variables

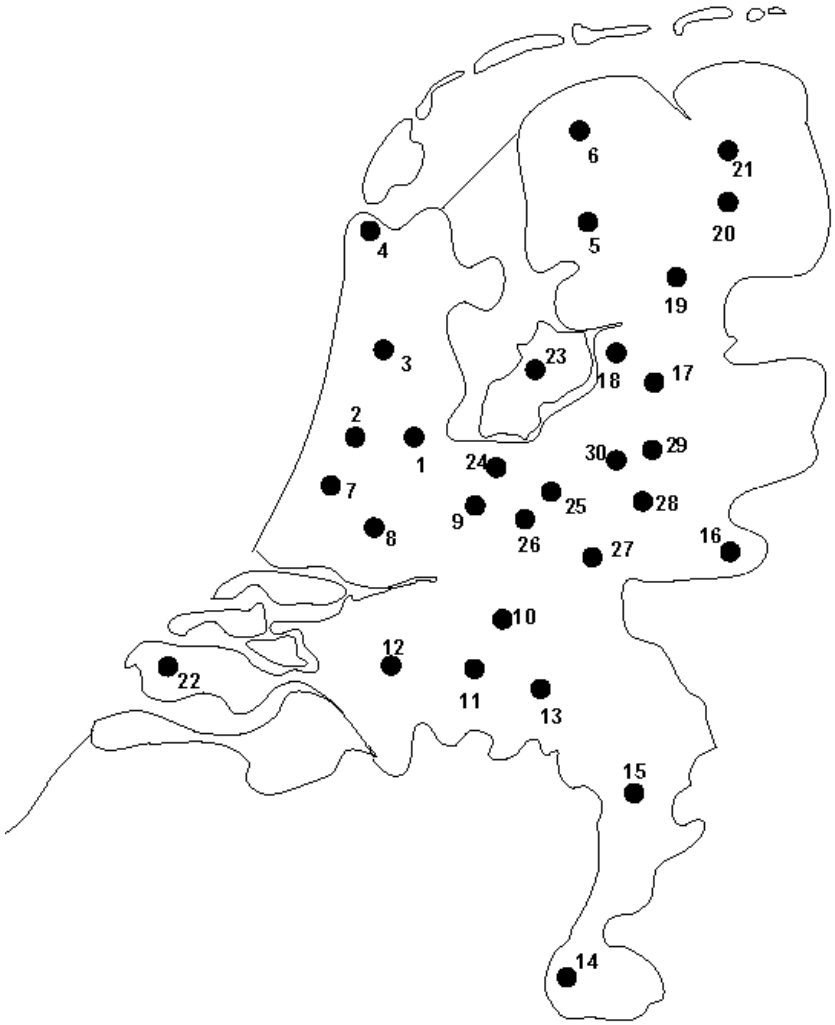
Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once

Permutation representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



ambrosys

Mutation operators for permutations

- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation : let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Insert Mutation for permutations

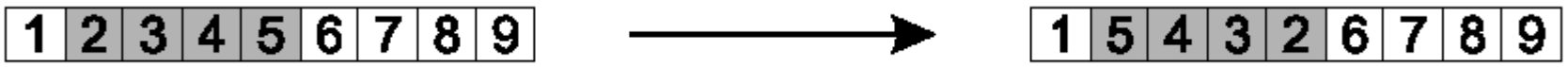
- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

Swap mutation for permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more

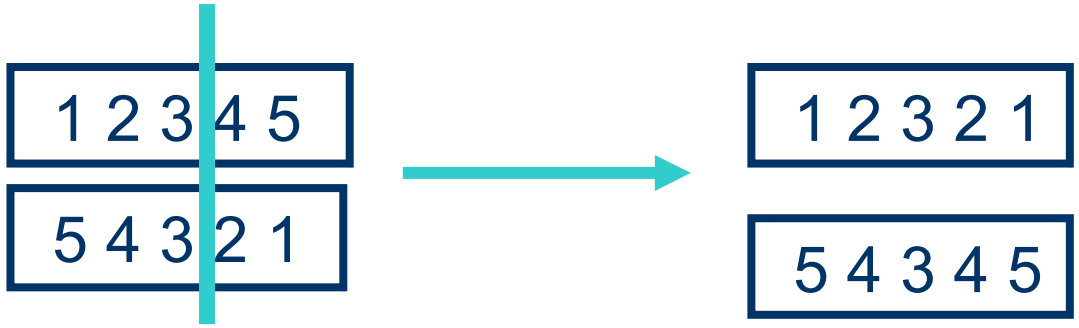
Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Crossover operators for permutations

- “Normal” crossover operators will often lead to inadmissible solutions



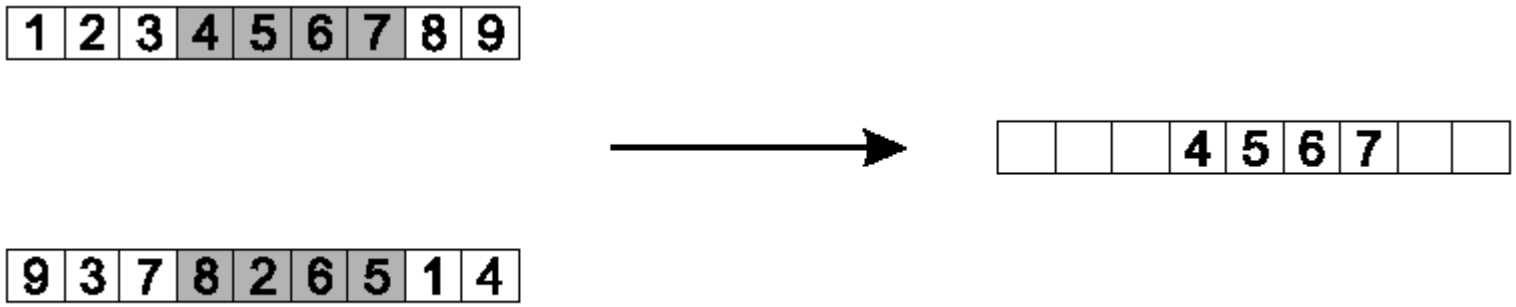
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order 1 crossover

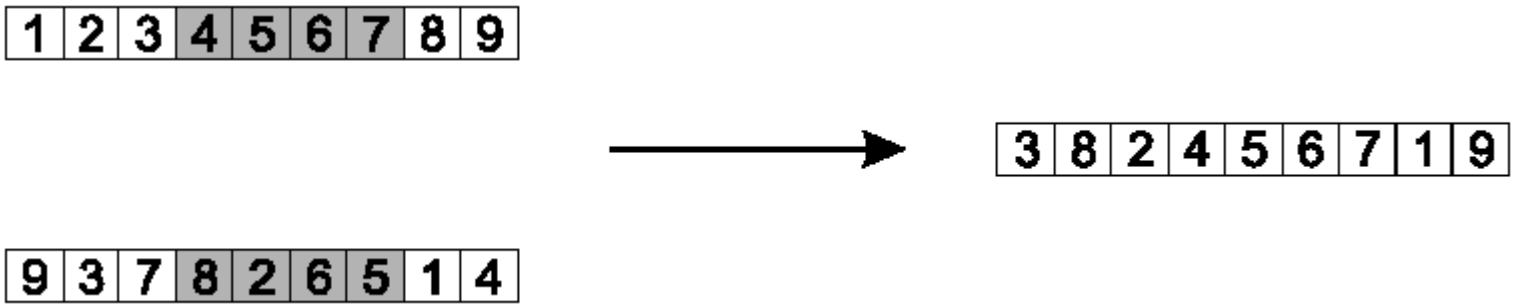
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order** of the second parent
     - and wrapping around at the end
  4. Analogous for the second child, with parent roles reversed

Order 1 crossover example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Cycle crossover

Basic idea:

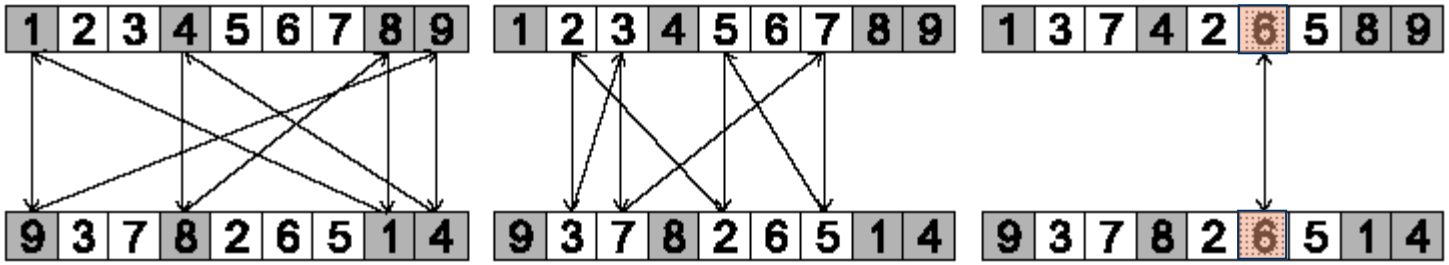
Each allele comes from one parent *together with its position*.

Informal procedure:

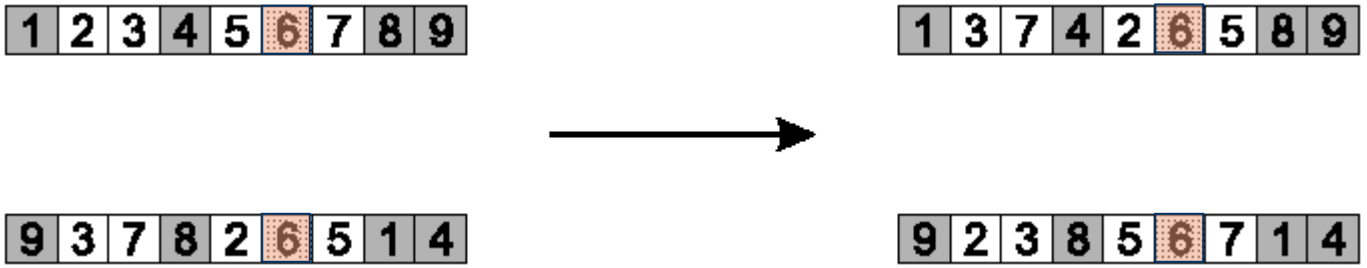
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Population Models

- SGA uses a Generational model:
  - each individual survives for exactly one generation
  - the entire set of parents is replaced by the offspring
- At the other end of the scale are Steady-State models:
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for GGA, 1/pop_size for SSGA

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
- Distinction between selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

Implementation example: SGA

- Expected number of copies of an individual *i*

$$E(n_i) = \mu \cdot f(i)/\bar{f}$$

(μ = pop.size, f(i) = fitness of i, $\bar{f}$ avg. fitness in pop.)

- Roulette wheel algorithm:
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections
  - No guarantees on actual value of n_i
- Baker's SUS algorithm:
  - *n* evenly spaced arms on wheel and spin once
  - Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$

Fitness-Proportionate Selection

- 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, lose selection pressure
- Scaling can fix last two problems
  - Windowing: $f'(i) = f(i) - \beta^t$
    - where β is worst fitness in this (last n) generations
  - Sigma Scaling: $f'(i) = max(f(i) - (\bar{f} - c \cdot \sigma_f), 0.0)$
    - where c is a constant, usually 2.0

Tournament Selection

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

Tournament Selection 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*

Two Special Cases

- Elitism
  - Widely used in both population models (GGA, SSGA)
  - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. “delete-worst”
  - From Whitley’s original Steady-State algorithm
  - Rapid takeover : use with large populations or “no duplicates” policy

Genetic Programming

GP quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - Competes (and most of the time beats) with neural nets and alike (SVM, decision trees...)
  - needs huge populations (thousands)
  - Slow, but acceptable on modern machines (couple of hours to couple of weeks for complex real life problems)
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)

GP technical summary tableau

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

Introductory example: credit scoring

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

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

Introductory example: credit scoring

- A possible model:

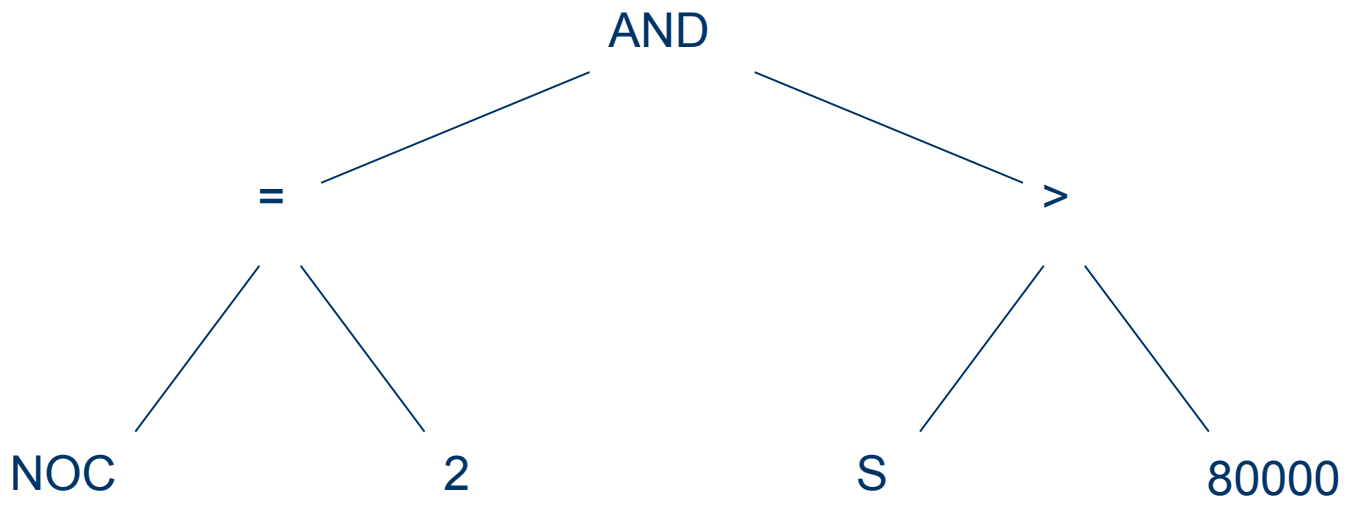
  IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
- In general:

  IF formula THEN good ELSE bad
- Only unknown is the right formula, hence
- 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
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

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

can be represented by the following tree

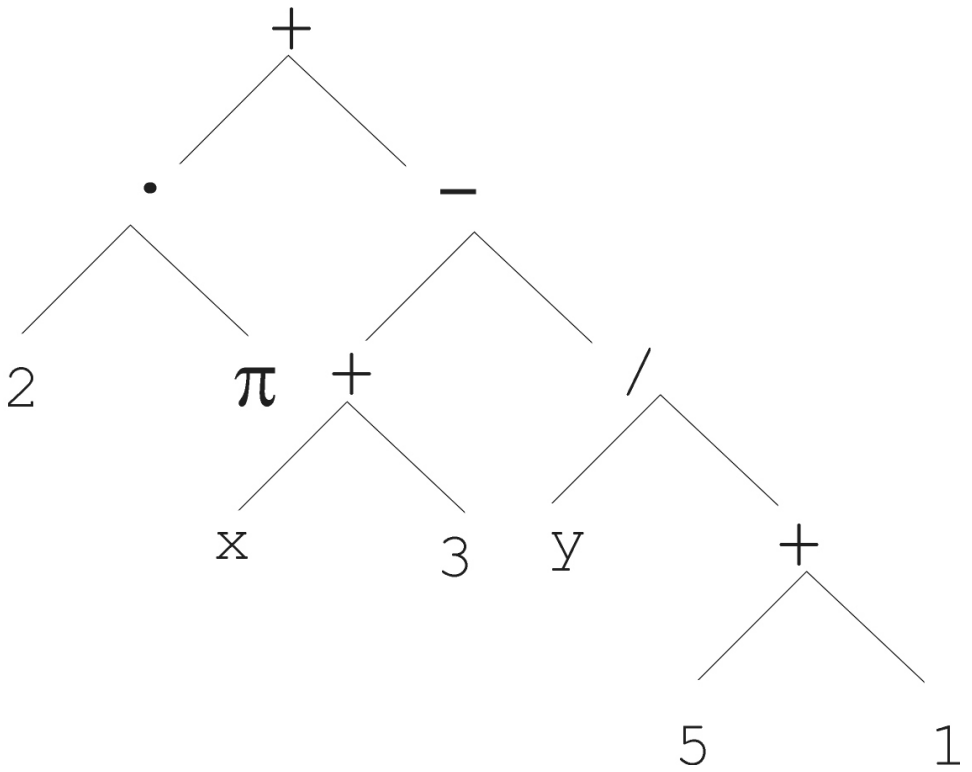


Tree based representation

- Trees are a universal form, e.g. consider
- Arithmetic formula $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program

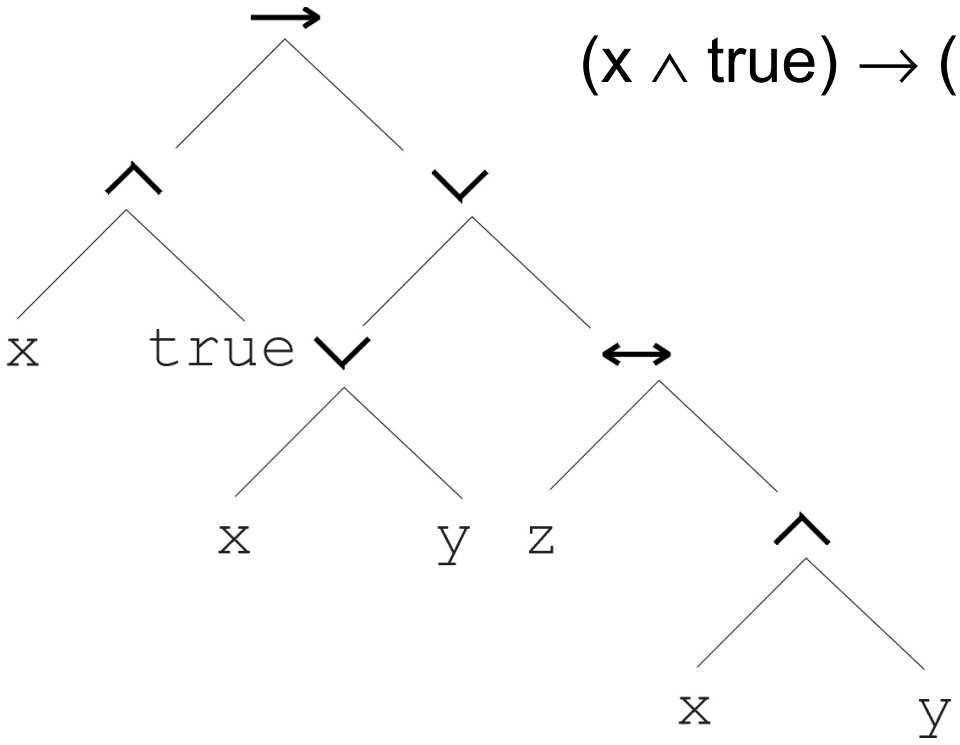
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



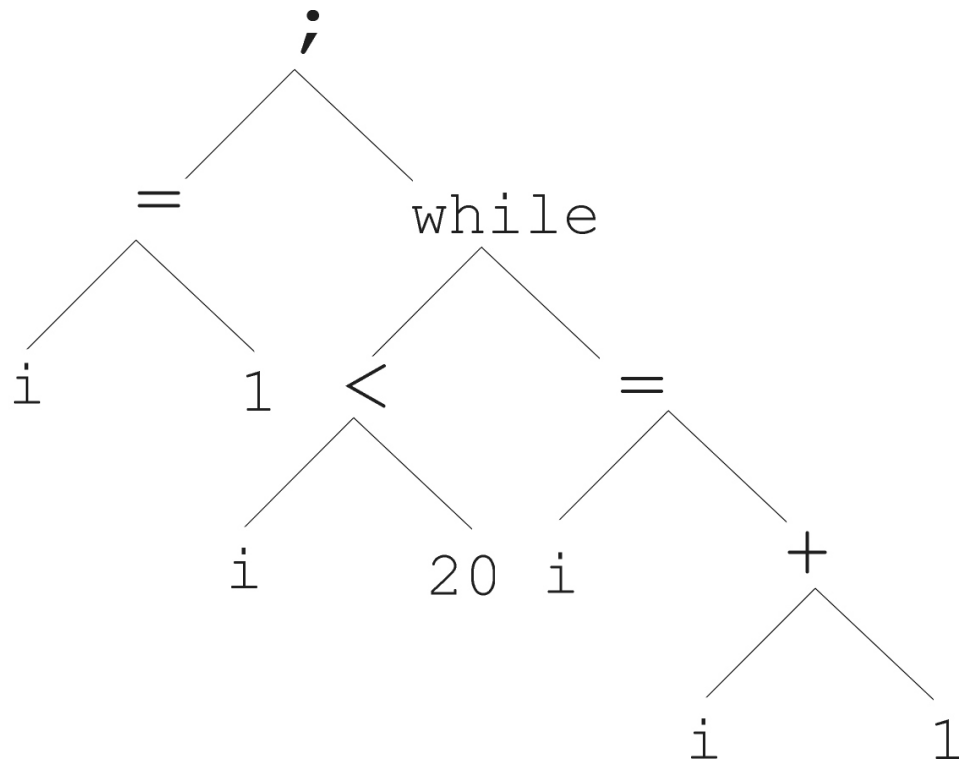
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree based representation



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

Tree based representation




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

Tree based representation

- 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

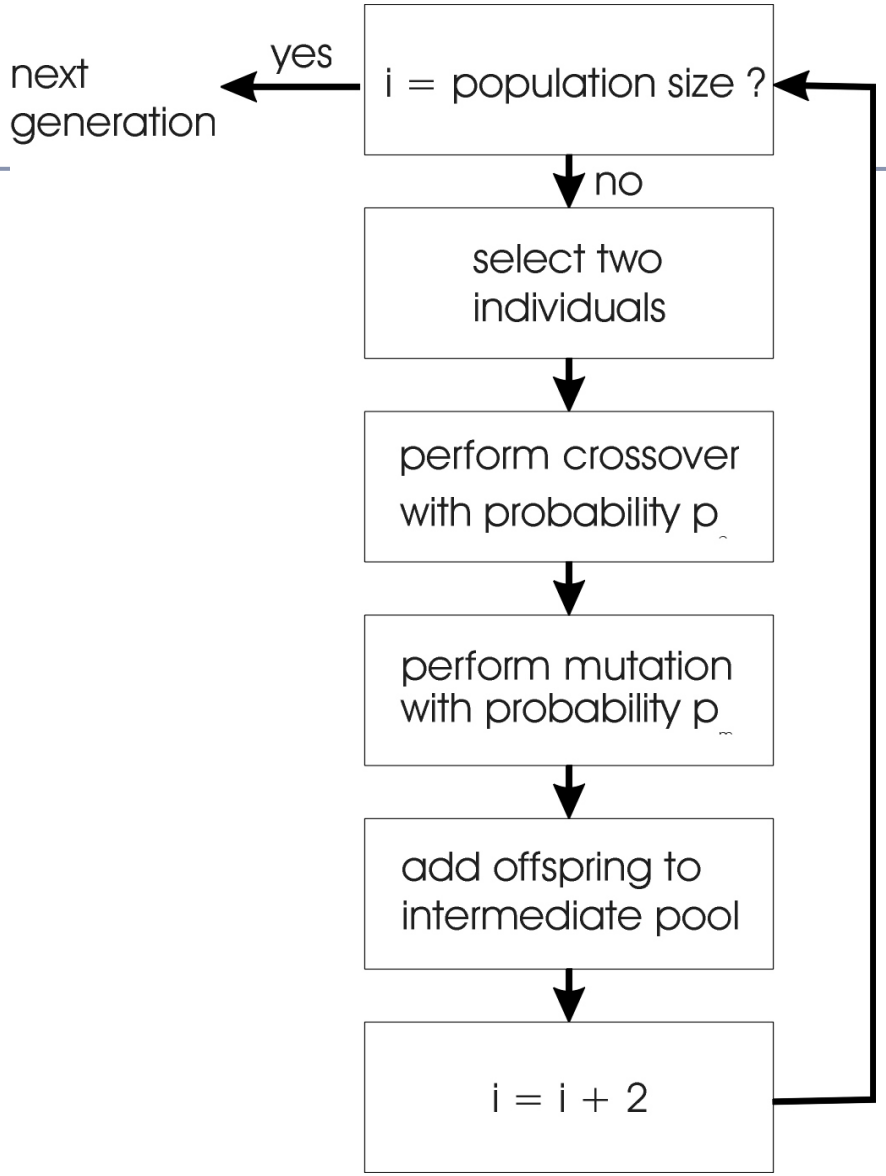
Tree based representation

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  1. Every $t \in T$ is a correct expression
  2. $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  3. There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

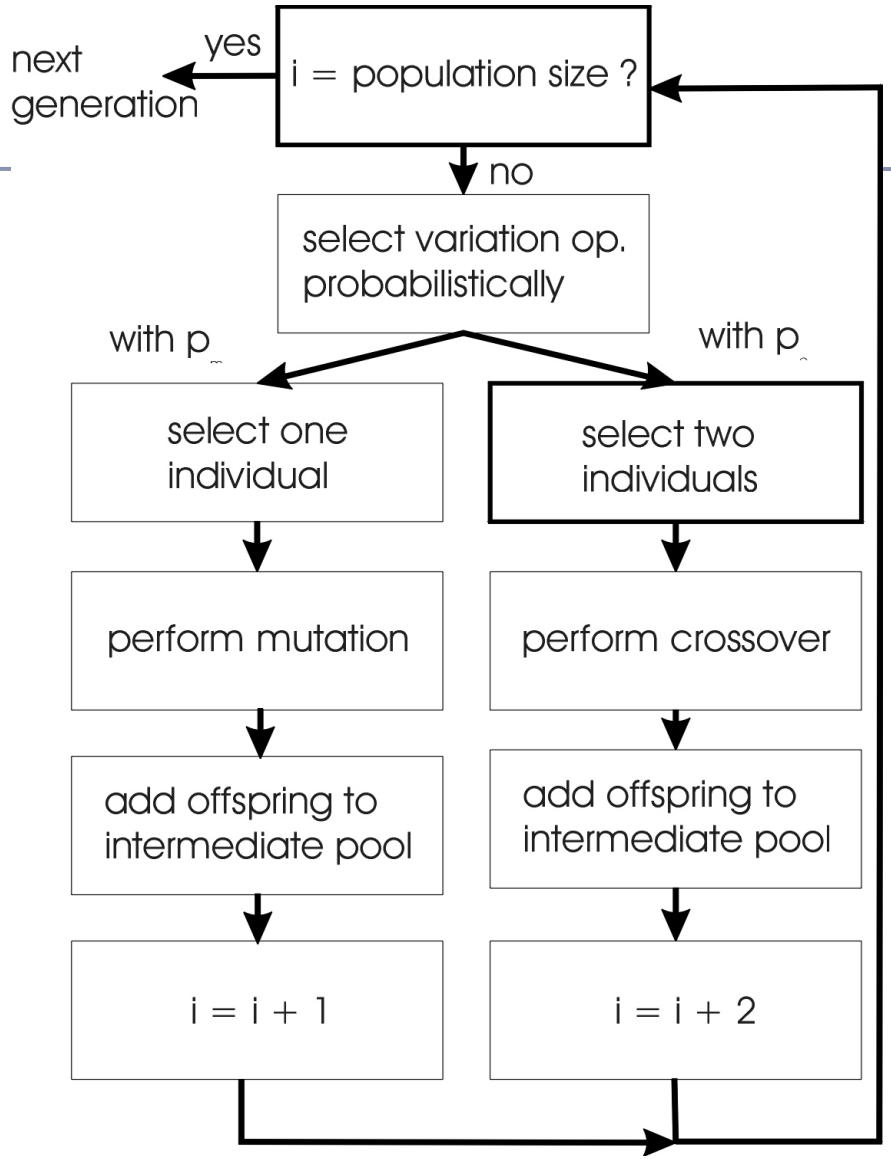
Offspring creation scheme

Compare

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



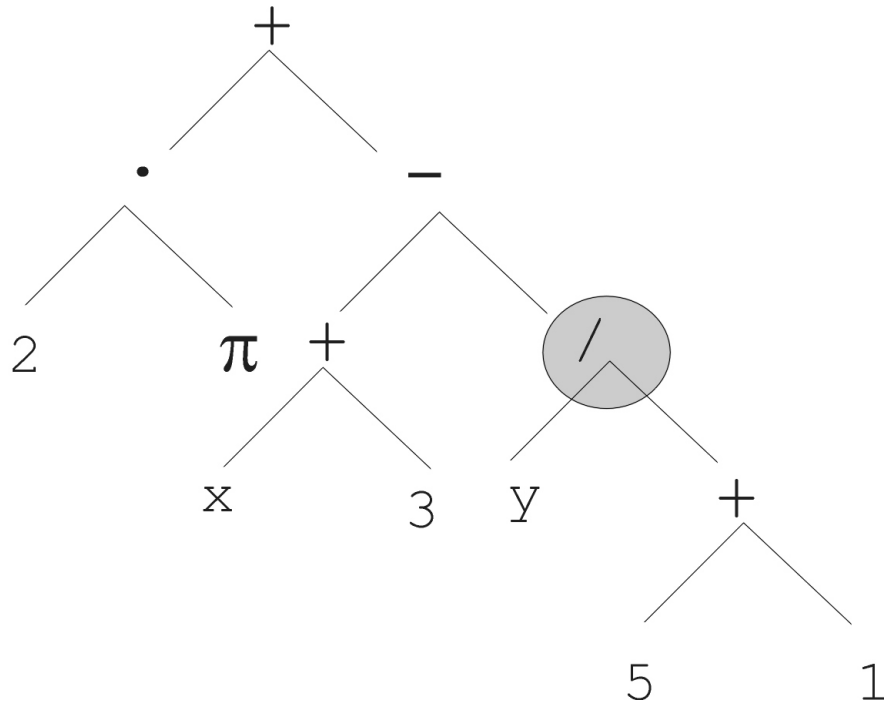
GA flowchart

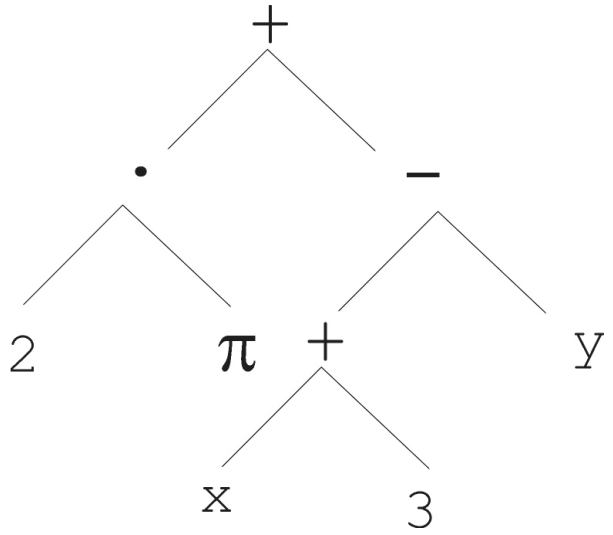


GP flowchart

Mutation

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



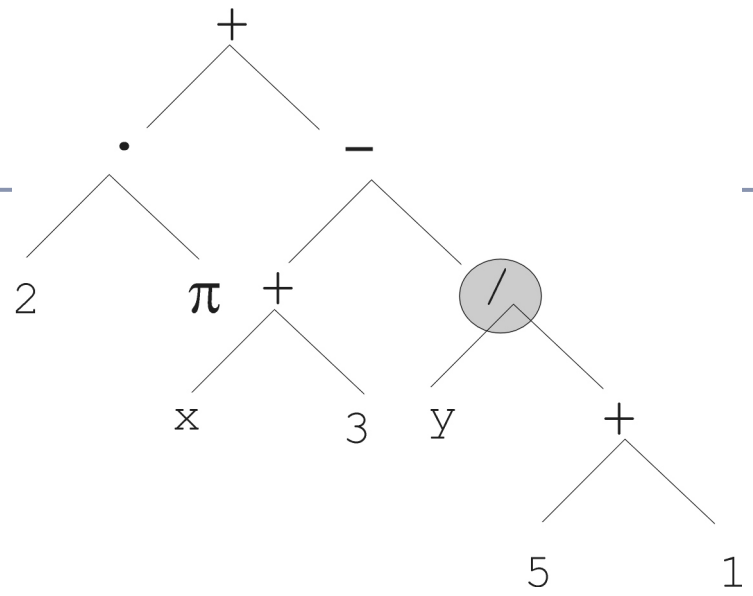


Mutation cont'd

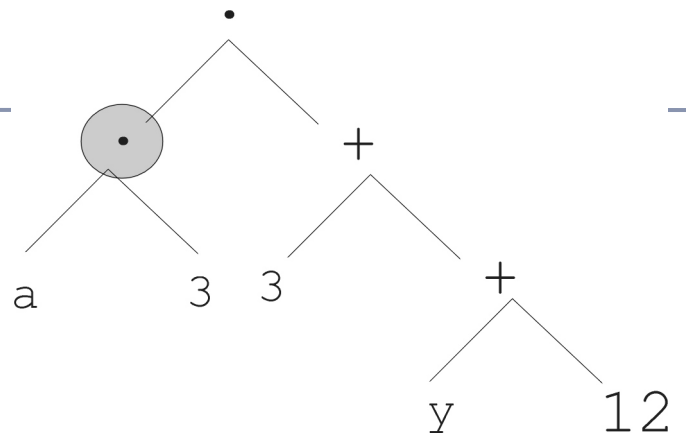
- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the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

Recombination

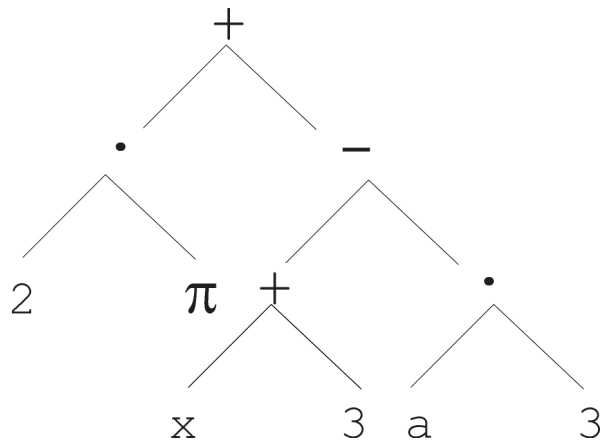
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of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



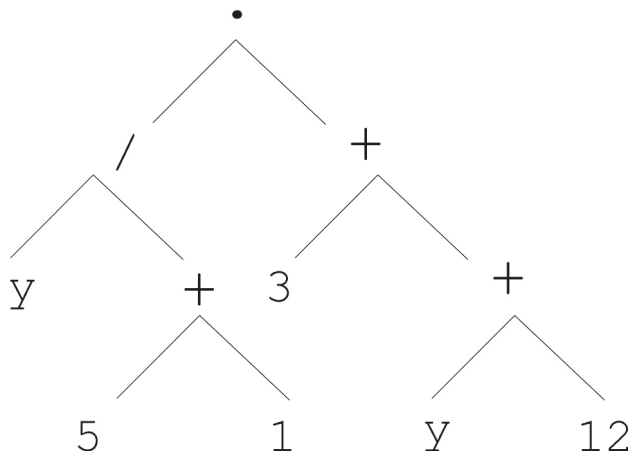
Parent 1



Parent 2



Child 1



Child 2

ambrosys

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%
- Survivor selection:
  - Typical: generational scheme (thus none)

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 $\leq D_{max}$):
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - 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

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

Problems involving “physical” environments

- Trees for data fitting vs. trees (programs) that are “really” executable
- Execution can change the environment → the calculation of fitness
- Example: robot controller
- Fitness calculations mostly by simulation, ranging from expensive to extremely expensive (in time)
- But evolved controllers are often to very good

Example application: symbolic regression

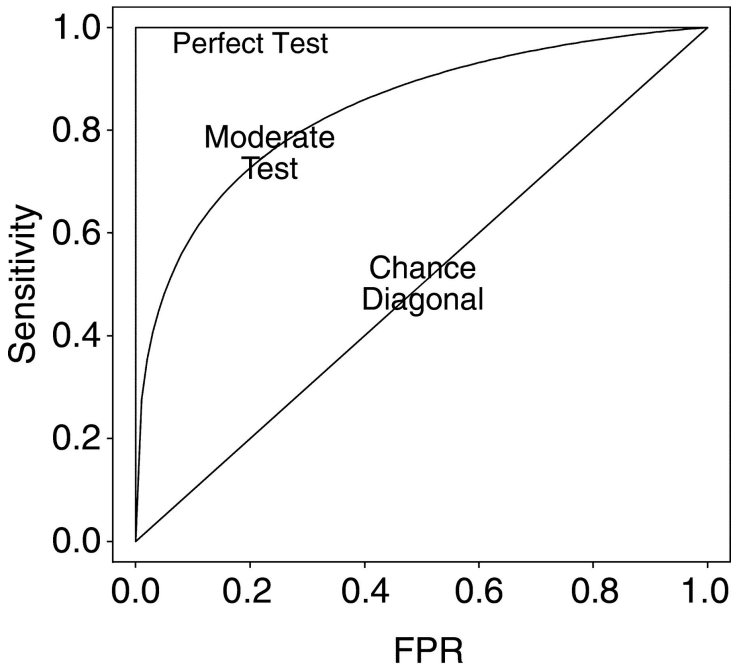
- Given some points in $\mathbf{R}^2$, $(x_1, y_1), \ldots, (x_n, y_n)$
- Find function f(x) s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- Possible GP solution:
  - Representation by F = {+, -, /, sin, cos}, T = $\mathbf{R} \cup \{x\}$
  - Fitness is the error $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - All operators standard
  - pop.size = 1000, ramped half-half initialisation
  - Termination: n "hits" or 50 generations reached (where "hit" is if $| f(x_i) - y_i | < 0.0001$)

Genetic Programming for Classification

- Genetic Programming can perform classification:
  - Use only binary trees: only works with very few problems
  - Use threshold on a regression problem
    - How to set the threshold?

Genetic Programming for Classification

- Fitness in classification: area under the ROC curve:
  - In signal detection theory, a receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied.



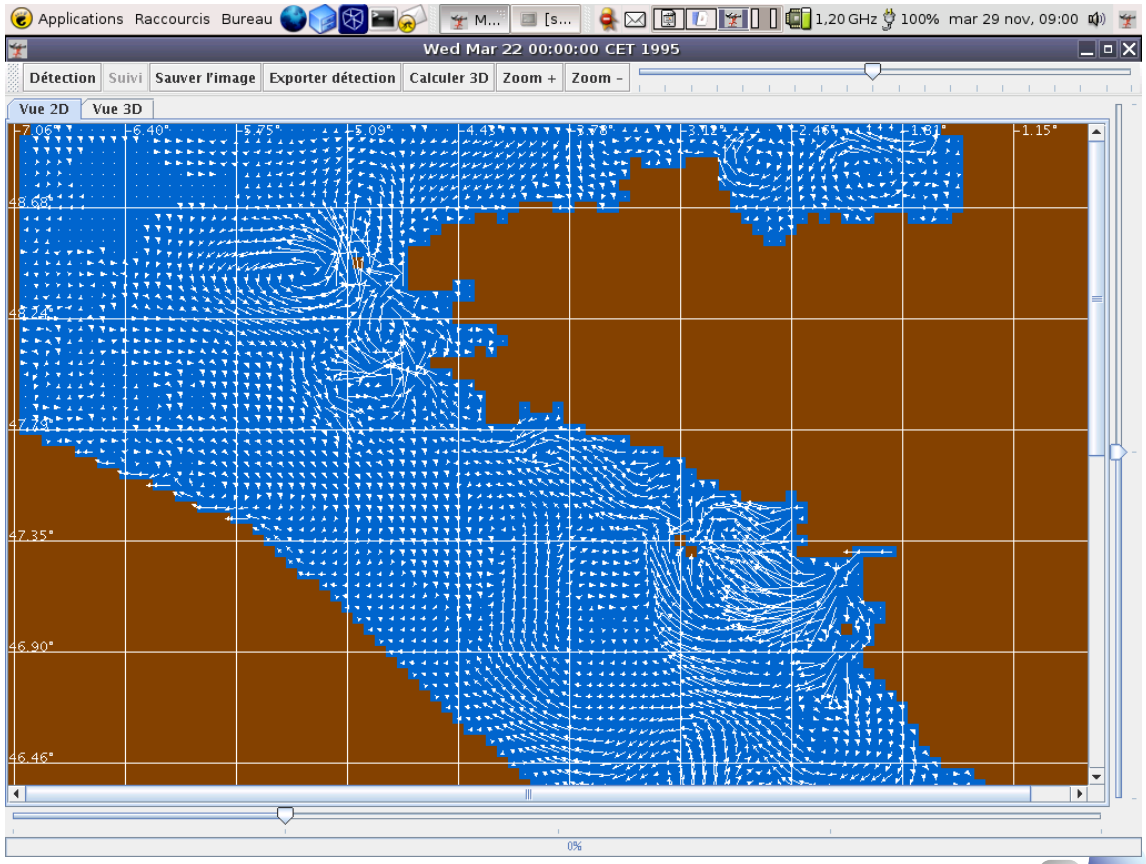
ambrosys

Genetic Programming for Classification

- The fitness function is in this case the maximization of the area under the ROC curve:
  - Gives an idea of the sensitivity and the classification error variation of the classifiers.
  - In the case that various « best » classifiers are kept, one is able to select the one that best fits to the needs of the problem
  - Makes possible the combination of various classifiers depending on the threshold we want to apply
  - Quite fast to compute O(nlog(n))

Genetic Programming for Classification

- Example: Genetic Programming for Retentive Structures Detection



ambrosys

Genetic Programming for Classification

- In this problem, we used a “standard” set of nodes (+,-,*,/,sin, cos), and added the problem specific inputs:
  - Strength
  - Rotational
  - Divergence
  - Strength 3x3
  - Angle 3x3
  - DistCoast

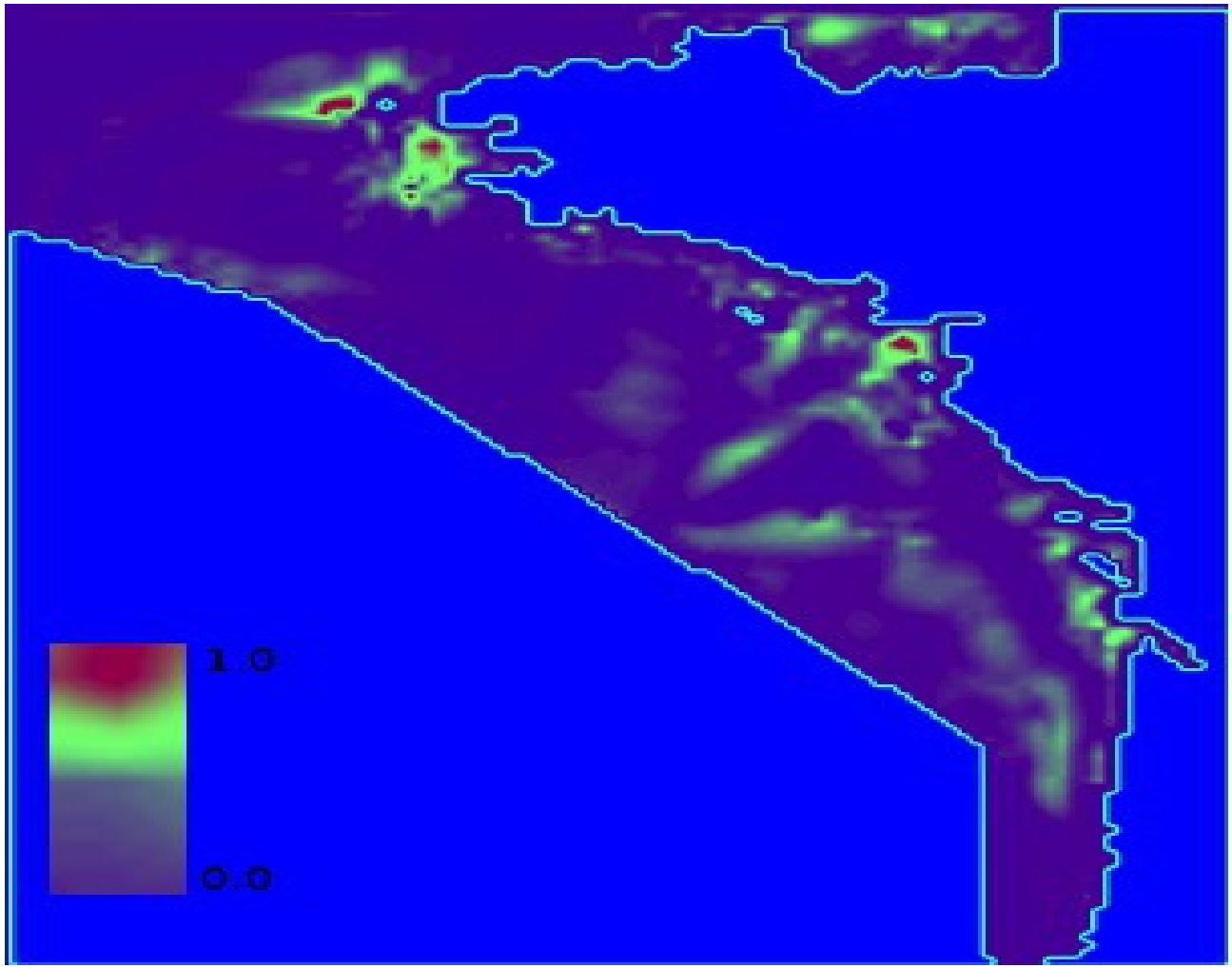
Genetic Programming for Classification

- An individual takes all the features of a cell as an input and returns a probability value.
- An individual performs the classification on every cell of a map and returns a probability matrix for the cells to belong to a retentive structure

Genetic Programming for Classification

- Parameters of the genetic programming algorithm:
  - 80 generations
  - 600 individuals
  - Max depth of a tree: 15
  - Mutation rate: 1%
  - Crossover rate: 94%
  - Reproduction rate: 5% with elitism
- Fitness: area under the ROC curve
- Learning time: more or less 1h
- Classification time: real time (for one map)

Genetic Programming for Classification



ambrosys

So what?

- Evolutionary Computation is very useful when deterministic methods fail or are not applicable
  - This is the case in many real life problems
- Genetic algorithms are very efficient in complex optimization problems (very huge search space), and can solve multi-objective optimization problems
- Genetic Programming is one of the best methods for multi-variable non linear function regression
  - This is also the case in many real life problem
- These techniques are widely successfully applied for industrial applications and are very flexible

More?

- Genetic Programming still has to spread to other communities
- Learning time is still quite high, and depends on the problem (most of the processing time is used for evaluation)
- But generated solutions are very often very efficient

How do I do?

- Several toolkits:
  - ECJ (java) includes GA, GP, ES and more.
    (http://cs.gmu.edu/~eclab/projects/ecj/)
  - OpenBeagle (C++) includes GA, GP.
    (http://code.google.com/p/beagle/)
  - Many more small toolkits (Python, MathLab, R...)

The End!

Thanks for your attention!