Evolutionary Algorithms

Oana Cocarascu & Helen Yannakoudakis

Department of Informatics
King's College London

Today

- Genetic Algorithms (GAs)
- Genetic Programming (GP)
  (Mitchell chapter 9)
- Co-evolutionary Algorithms
- Examples

KING'S College LONDON

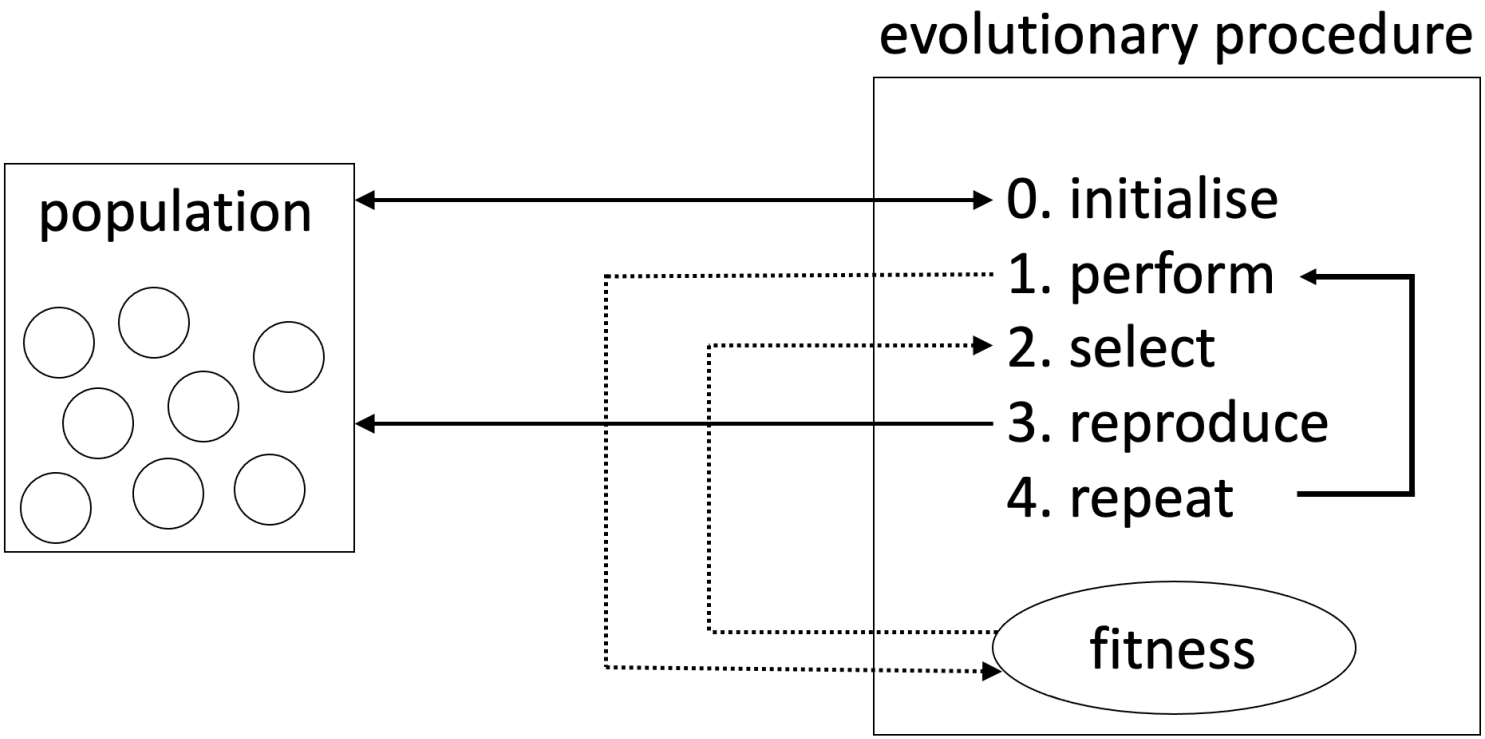
Evolutionary Learning

- An evolutionary algorithm (EA) is a way of doing search.
- Inspired by evolutionary biology and concepts from genetics.
- We have a "population" of individuals where each "individual" is a representation of a solution to a problem.
- We build a representation and decide how to combine these representations by doing:
  - selection; and
  - reproduction.
- Establishing and evaluating a population can be viewed as a (massively) parallel search.

KING'S College LONDON

Evolutionary Learning

- Simulation of the evolution process.
  - A population of candidate solutions evolves towards better solutions by repeatedly selecting the fit individuals from the current population and modifying them to form the next generation population.



Genetic Algorithms (GAs)

- Introduced by John Holland (1962, 1975)
- Overview
- Advantages
- The basic algorithm
- The basic components
- Trade-offs

GAs: Overview

- Loosely based on the idea of simulated evolution: "survival of the fittest" (Darwin).
- In general, we hypothesise a population of candidate solutions to a particular problem.
- Then we evaluate each member of the population to decide how good that candidate is at solving the problem.
- Then we select those members of the population that are the best (the "fittest").
- And we reproduce to obtain new members of the population.
- Then we start all over again, hypothesising with this new set of candidate solutions.

KING'S College LONDON

GAs: Advantages

- Good for situations where it is difficult to ascertain the impact of a particular partial solution to a problem.
- Easy to implement through parallel processing, because each candidate solution can be evaluated on its own.
- Can adapt easily in dynamic environments.

GAs: The Basic Algorithm

GA(*Fitness*, *Fitness_threshold*, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.
Fitness_threshold: A threshold specifying the termination criterion.
p: The number of hypotheses to be included in the population.
r: The fraction of the population to be replaced by Crossover at each step.
m: The mutation rate.

- *Initialize:* $P \leftarrow p$ random hypotheses
- *Evaluate:* for each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$
  1. *Select*
  2. *Crossover*
  3. *Mutate*
  4. *Update*
  5. *Evaluate*
- Return the hypothesis from P that has the highest fitness.

KING'S College LONDON

GAs: The Basic Algorithm

GA(*Fitness*, *Fitness_threshold*, p, r, m)

- *Initialize:* $P \leftarrow p$ random hypotheses
- *Evaluate:* for each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$
  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S.

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Choose m percent of the members of P_s, with uniform probability. For each, invert one randomly selected bit in its representation.
  4. *Update:* $P \leftarrow P_s$.
  5. *Evaluate:* for each h in P, compute $Fitness(h)$.
- Return the hypothesis from P that has the highest fitness.

KING'S College LONDON

GAs: Components of the basic algorithm

- Representation - how is each candidate represented?
- Fitness - how is each candidate evaluated?
- Selection - how are the best candidates chosen?
- Reproduction - how are new candidates generated?

GAs: Representation

- In a Genetic Algorithm, the classic representation is a binary string (or “bit string”).
- Each element in the string represents the presence (or absence) of a particular “trait”.
- The bit string is the genotype.
- The set of traits encoded is the phenotype.
- **Representation is domain dependent.**
- *Note that biologists generally do not like this terminology because it only very loosely resembles real genetics.*

KING'S College LONDON

GAs: Representation

- Example:

Table 1.2 The weather data.

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

[WFH Table 1.2]

GAs: Representation

0	play	yes/**no**		→	true/**false**
1	windy	true/**false**		→	true/**false**
2	outlook	**sunny**		outlook-sunny	**true**/false
3		overcast	→	outlook-overcast	true/**false**
4		rainy		outlook-rainy	true/**false**
5	temp.	**hot**		temp-hot	**true**/false
6		cool	→	temp-cool	true/**false**
7		mild		temp-mild	true/**false**
8	humidity	**high**	→	humid-high	**true**/false
9		normal		humid-normal	true/**false**

- becomes a 10-bit string:

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	1	0

GAs: Representation

- Note that it is possible to generate genotypes that have:
  - Multiple phenotypes
  - Uncertain phenotypes
  - Invalid phenotypes
- Different strategies exist for dealing with the situation of invalid phenotype, such as:
  - Preventing generation of invalid phenotypes.
  - Reporting lowest fitness for invalid phenotypes.

KING'S College LONDON

GAs: Representation

- For example:

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	**1**	**1**

- Could be interpreted in different ways, as follows:
  - Multiple phenotypes $\Rightarrow$ rule is certain: it applies to 2 cases: don't play (bit 0 is **0**) when humidity is either high or normal.
  - Uncertain phenotype $\Rightarrow$ rule is uncertain: humidity could be either high or normal and we do not know which (but don't play in either case).
  - Invalid phenotype $\Rightarrow$ rule is invalid: humidity cannot be high and normal at the same time.

GAs: Fitness

- Fitness = metric of success.
- Similar to a scoring mechanism, it is a way of measuring how well a rule (or algorithm) performs.
- Fitness is measured for each candidate solution in a population.
  - 100 individuals in a population means 100 fitness values.
- Examples:
  - Accuracy of a function given training data.
  - Win rate in a game playing algorithm.
  - Speed for a robot walking algorithm.
- **Fitness is domain dependent**.

KING'S College LONDON

GAs: Selection

- Classic method of selection is probabilistic:

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{N} Fitness(h_j)}$$

  where
  - N is the size of the population
  - Each h is the representation of a candidate solution
  - $Fitness(h)$ is the fitness (score) of that candidate
- This method is called fitness proportionate selection.
- Also called roulette wheel selection.
- **Fitness proportionate selection is domain independent.**

KING'S College LONDON

GAs: Selection

- Other methods of selection:
- Tournament selection:
  - Randomly select two candidates from the population.
  - Keep the candidate with the higher fitness, with probability p proportional to each candidate's fitness (i.e., the more fit candidate is more likely to survive).
- Rank selection:
  - Sort candidates and rank them according to their fitness.
  - Keep candidates with rank higher than (or equal to) proportion p.
- **Tournament selection and rank selection are domain independent.**

KING'S College LONDON

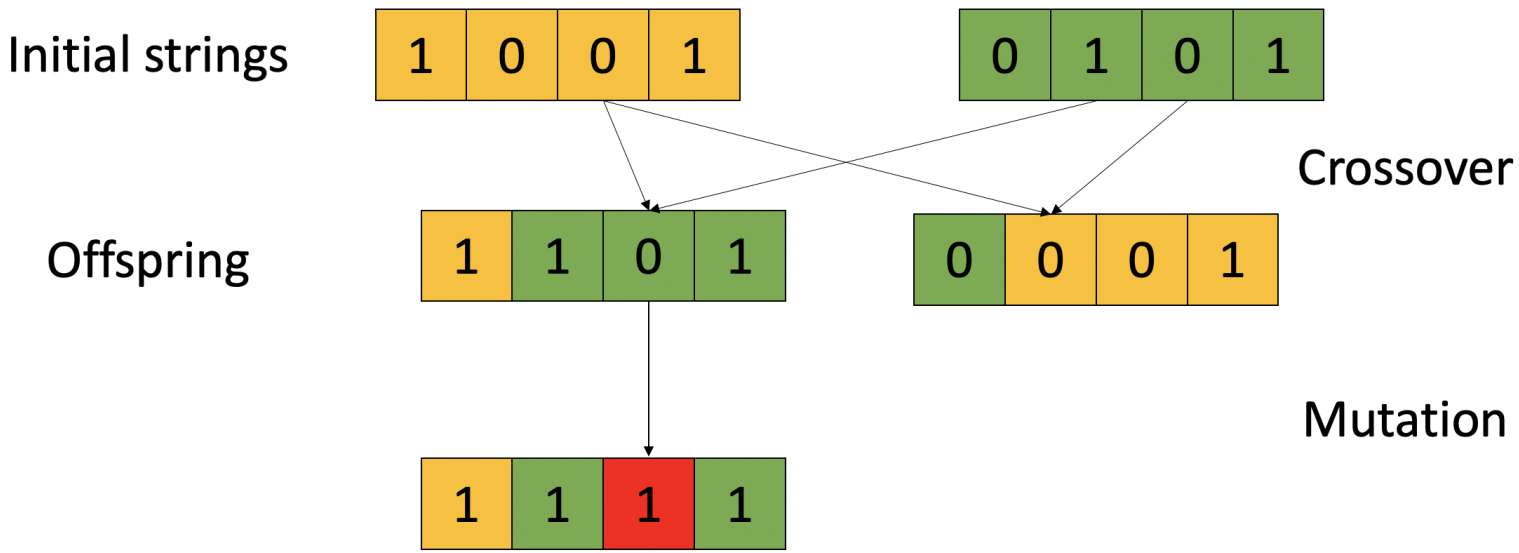
GAs: Selection

- In the classic algorithm, a fixed fitness threshold is defined.
- **Fixed threshold value is domain dependent**.
- We segment the population of N candidates into 2 groups:
  - P_{keep} = candidates whose fitness is $\geqslant$ threshold.
  - $P_{replace}$ = candidates whose fitness is $<$ threshold.
- Probabilistic selection is applied to P_{keep}.
  - Fitness proportionate: randomly select candidates from P_{keep}.
  - Rank: select top p from P_{keep}.
  - Tournament: randomly select two candidates to compare from P_{keep}.
- The proportion is known as the exploration-exploitation ratio:
  - $P_{keep} \Rightarrow$ exploitation.
  - $P_{replace} \Rightarrow$ exploration.

KING'S College LONDON

GAs: Reproduction

- Two main reproduction operators are:
  - Crossover
  - Mutation



GAs: Reproduction

- In classic algorithm, where population size = N
  - Crossover rate = r, where $0 \leqslant r \leqslant 1$
  - Mutation rate = m, where $0 \leqslant m \leqslant 1$
- New population is generated with two steps:

first $\rightarrow$ $\boxed{\begin{array}{c}(r \cdot N) \\ \text{crossed over}\end{array}} + \boxed{\begin{array}{c}((1-r) \cdot N) \\ \text{selected as is}\end{array}}$

second $\rightarrow$ $\boxed{\begin{array}{c}(m \cdot N) \\ \text{mutated}\end{array}} + \boxed{\begin{array}{c}((1-m) \cdot N) \\ \text{selected as is}\end{array}}$

KING'S College LONDON

GAs: Reproduction - Crossover

- 1-point crossover
  - Randomly select one bit in two strings:

parent1	0	0	1	0	0	**1**	0	0	1	1
parent2	1	1	1	0	0	**0**	1	0	0	0

  - and swap bits from that point:

offspring1	0	0	1	0	0	0	1	0	0	0
offspring2	1	1	1	0	0	1	0	0	1	1

- *n*-point crossover: select n points, $n \leqslant$ length of the string
  - Higher values of n are not very useful, there is too much variation or not enough as n approaches length of string.

GAs: Reproduction - Crossover

- A common way to specify crossover points is to use a crossover mask which indicates which bits should come from which parent.
- In our previous example, we would have:

mask	1	1	1	1	1	0	0	0	0	0
parent1	0	0	1	0	0	1	0	0	1	1
parent2	1	1	1	0	0	0	1	0	0	0

- Produces same result as on previous slide:

offspring1	0	0	1	0	0	0	1	0	0	0
offspring2	1	1	1	0	0	1	0	0	1	1

GAs: Reproduction - Crossover

- Uniform crossover
  - Create mask by randomly selecting bits from each parent, using a uniform distribution.
  - Thus, it is equally likely to draw any bit from either parent.
- Other distributions can be used, for example if you want to weigh certain bits differently from others.

GAs: Reproduction - Mutation

- 1-point mutation
  - Randomly select one bit in one string:

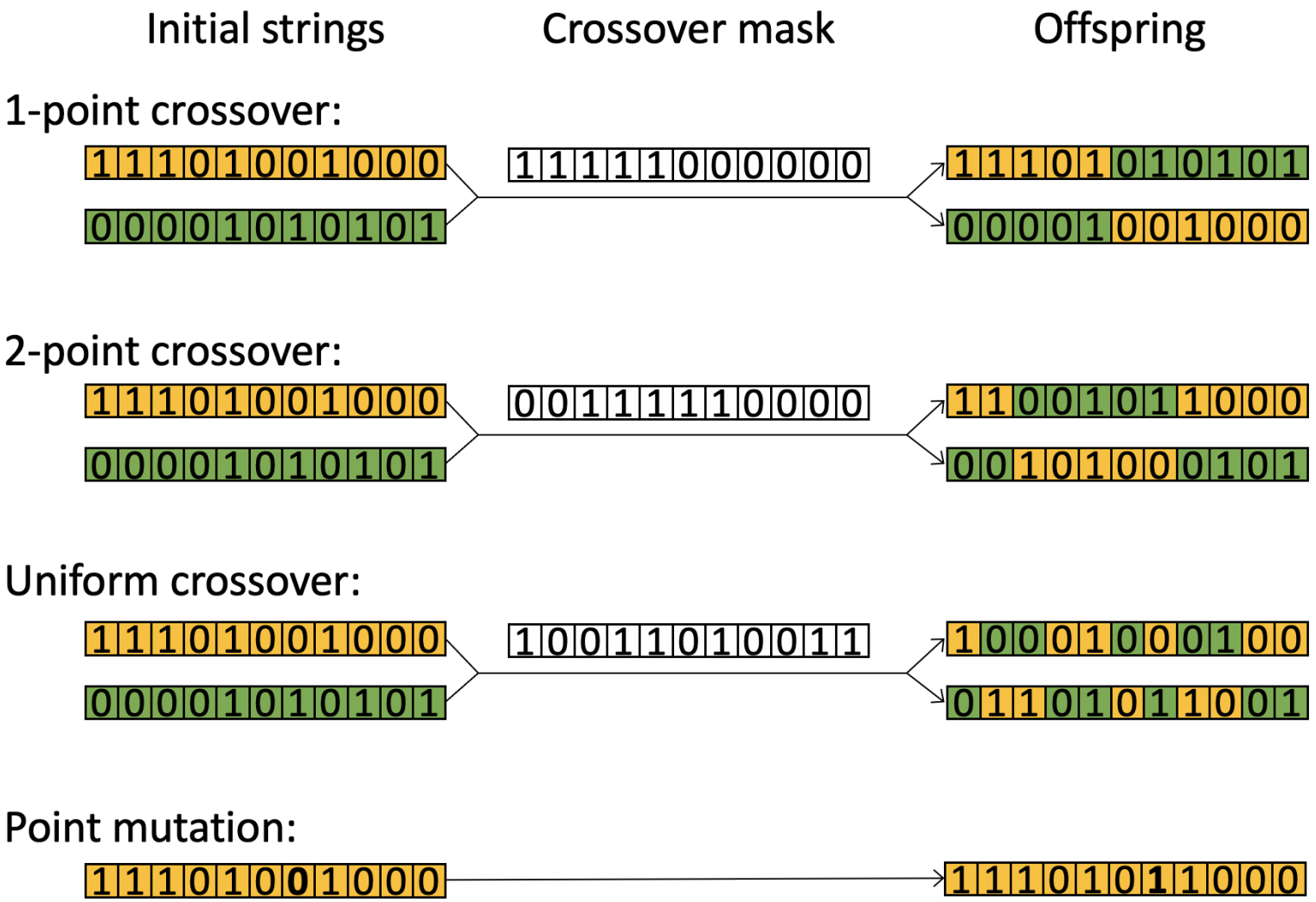
0	0	1	0	0	**1**	0	0	1	1

  - and flip it:

0	0	1	0	0	**0**	0	0	1	1

- *n*-point mutation: select *n* points, $n \leqslant$ length of the string.

GAs: Reproduction



GAs: Trade-offs

- Classic **trade-offs** when designing genetic algorithms are between the population size and the number of generations (iterations).
- A smaller population generally implies more generations are required;

  but each generation is evaluated relatively quickly.
- A larger population can mean that fewer generations are required;

  but each generation can take longer to be evaluated.

GAs: Trade-offs

- Exploration-exploitation ratio.
- A higher exploitation ratio means that the set of candidates moves around the solution space (also called *landscape*) more slowly, more smoothly (without jumping around too much from one generation to the next).
  - Solutions in a narrower search space are considered.
  - Search is conducted at a finer resolution.
- A higher exploration ratio means that the set of candidates jumps around the solution space.
  - Solutions at more disparate points in the search space are considered.
  - Search is conducted at a coarser resolution.

GAs: Trade-offs

- Nature vs nurture.
- In biology, there is an old debate about whether learnt behaviours are passed from parent to child in their genetic makeup, dating back to Lamarck (late 1800's).
- Current biological evidence disavows this view.
- Computational methods have shown that GAs/GP which learn and pass the learnt information to their offspring can exhibit improved results over parents that do not adapt.
- One example is the Baldwin effect:
  - If the environment is changing, then individuals that can adapt to the changes will survive longer.
  - Individuals that learn more are comprised, initially, of incomplete genetic code, which fills in as they learn.
- *Note that the inherent adaptivity in an EA is in the changing population (the change in membership) rather than changes in individual members.*

GAs: Variations

- Real-valued GA.
- Instead of bit string, string contains real numbers.
- Each number is an actual feature value (i.e., the value of the trait) instead of representing the presence or absence of a trait.
- **Mutation** randomly changes value (because there is no notion of “flipping”).
- Everything else works as with bit-string representations.

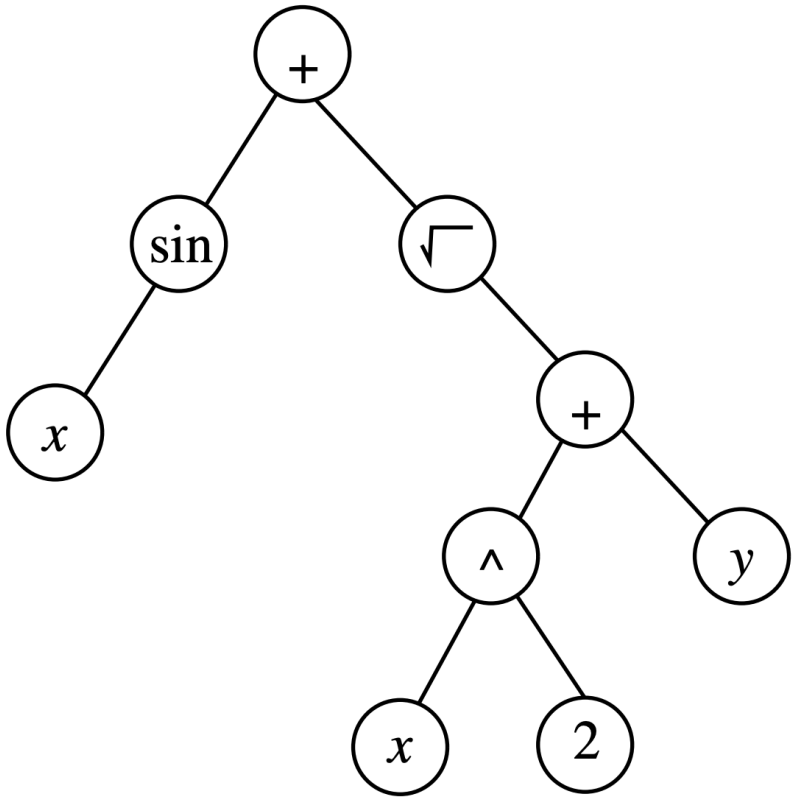
KING'S College LONDON

Genetic Programming (GP)

- Introduced by John Koza (1992).
- Key difference with GAs is the representation.
- GAs genotype: Binary string of fixed size.
- GP genotype: Program represented as tree.
- LISP s-expression.

KING'S College LONDON

GP: Representation



[Mitchell Figure 9.1]

- S-expression: *(+ (sin x) (sqrt (+ (pow x 2) y)))*
- Equation:

$$= sin\ x + \sqrt{x^2 + y}$$

GPs: Fitness

- Fitness is determined by running the program:

$$\text{fitness} = \textit{sin}\ x + \sqrt{x^2 + y}$$

- And then fitness (i.e., value returned by program) can be used directly:

$$\textit{if}\ (\text{fitness} \geqslant \text{fitness\_threshold}) :$$

to determine if candidate program belongs to

P_{keep}, or

$P_{replace}$

GPs: Selection

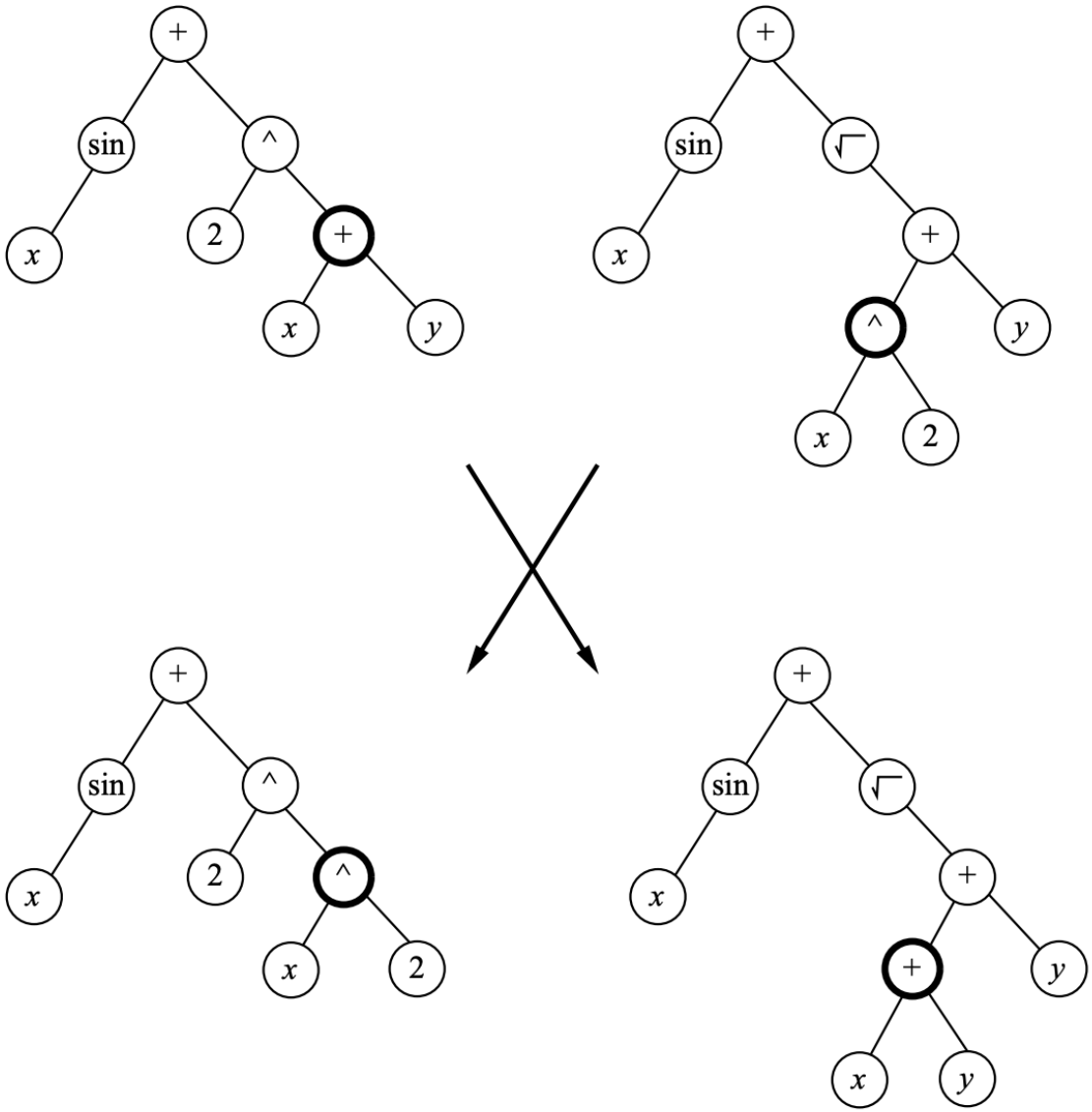
- Selection is performed in the same way as with GAs.
- Fitness proportionate selection.
- Rank selection.
- Tournament selection.

GPs: Reproduction

- Koza did not use mutation in his original work.
- But he did use crossover.
- With crossover, one (or more) nodes in the tree are selected from two parents;
- And then the sub-trees starting with those nodes are swapped.

KING'S College LONDON

GPs: Reproduction - Crossover



[Mitchell Figure 9.2]

GPs: Reproduction - Crossover

- parent1:

$$sin\ x + 2^{\mathbf{x}+\mathbf{y}}$$

- parent2:

$$sin\ x + \sqrt{\mathbf{x^2} + y}$$

- offspring1:

$$sin\ x + 2^{x^2}$$

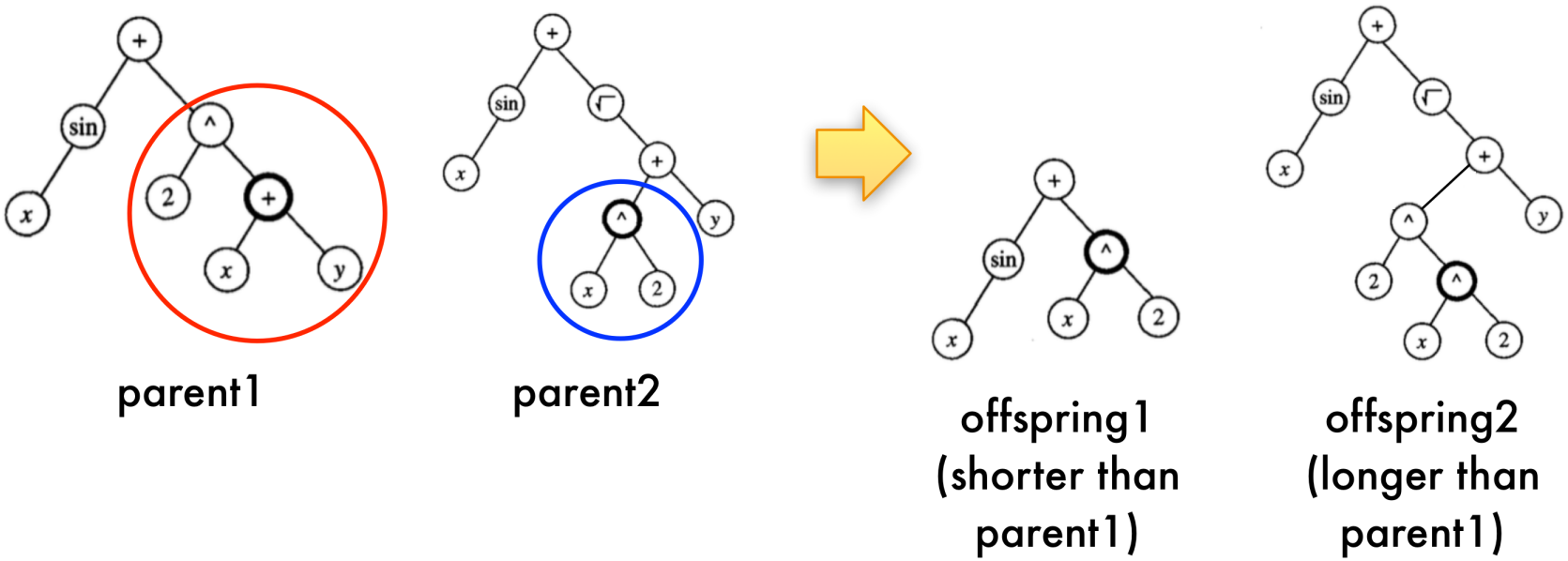
- offspring2:

$$sin\ x + \sqrt{(x + y) + y}$$

KING'S College LONDON

GPs: Code Bloat

- One significant issue with GP is code bloat.
  - The size of evolving programs grows extremely large (i.e., the length of the s-expression).
- Consider the modified example from slide 36:



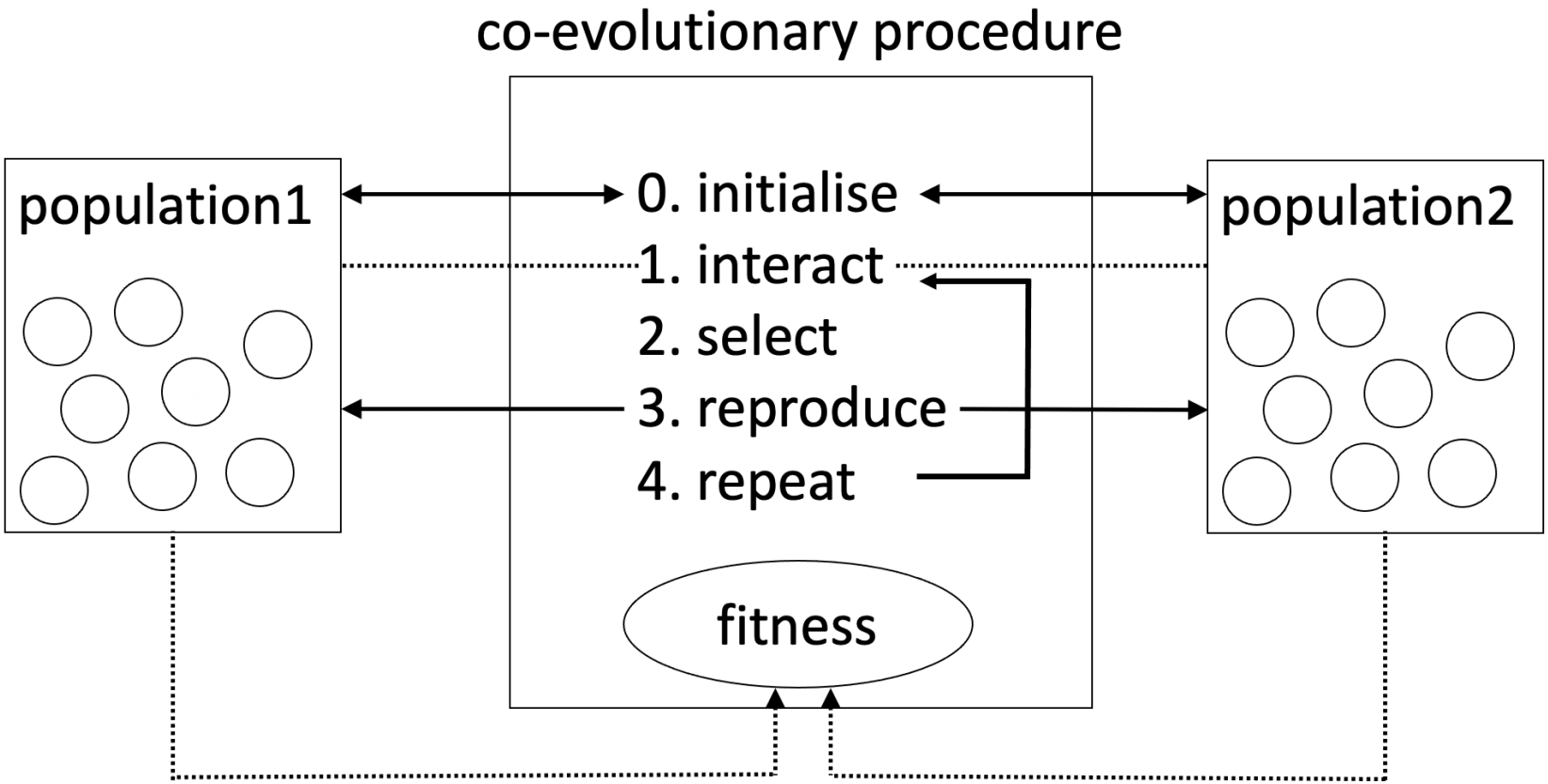
- Possible solution: limit the maximum length of an s-expression that results from crossover.

Co-evolution

- With traditional evolutionary algorithms, we have **one evolving population**.
- The success of candidate solutions in the population is measured against a fixed fitness function and a fitness threshold.
- With co-evolution, there are **two evolving populations**.
- The fitness of a candidate solution is measured by comparing members of opposing populations.
- The result is an arm's race, where one population (as a whole) reaches to surpass the other population.

KING'S College LONDON

Co-evolutionary Learning



Co-evolution

- There are distinct advantages with co-evolution when there is no easy way to define a fixed fitness function, or one simply does not exist.
- There still has to be a way to compare two solutions and decide between them which is better.
- Game playing is a classic example, where two candidate solutions play games against each other and the candidate that wins more games is declared the fittest.

Co-evolution: Basic components

- Representation: could be a GA or a GP.
- Fitness: members of opposing populations are compared against each other.
- Selection: same as evolution.
- Reproduction: same as evolution, but parents are chosen within the same population.

Co-evolution: Issues

- Collusion
  - When members of a population “live and let live”.
  - Effectively: nobody tries to win.
  - So each keeps getting better, but nobody becomes the best.
- Mediocre Stable State (MSS)
  - Also called suboptimal equilibria.
  - Where the populations settle into a portion of the landscape and do not change $\Rightarrow$ convergence.
  - But this might not be the optimal portion of the landscape.

KING'S College LONDON

Examples

1. Function with one parameter
2. Function with several parameters
3. Wall-following robot [Nilsson]

Function with one parameter

- Maximise $f(x) = x^2, 0 \leqslant x \leqslant 31$
- We know $f(x) = x^2$ has maximum value 961 when $x = 31$.
- Representation: 5 bits (11111 $\rightarrow$ 31).
- Fitness function: $f(x)$.
- Example of initial population where the population size is 4.

#	Individual h	x	$f(x)$	$Pr(h)$
1	01101	13	169	0.14
2	11000	24	576	0.49
3	01000	8	64	0.06
4	10011	19	361	0.31
		sum	1170	

KING'S College LONDON

Function with one parameter

- Selection
  - Roulette wheel which gives: #1, #2, #2, #4.
- Reproduction operators
  - Crossover
  - Mutation

Initial	x	$f(x)$	Parents	Crossover	Mutation	x	$f(x)$
01101	13	169	0110|1	01100	01100	12	144
11000	24	576	1100|0	11001	11001	25	625
01000	8	64	11|000	11011	11011	27	729
10011	19	361	10|011	10000	10010	18	324
	sum	1170				sum	1822

Function with several parameters

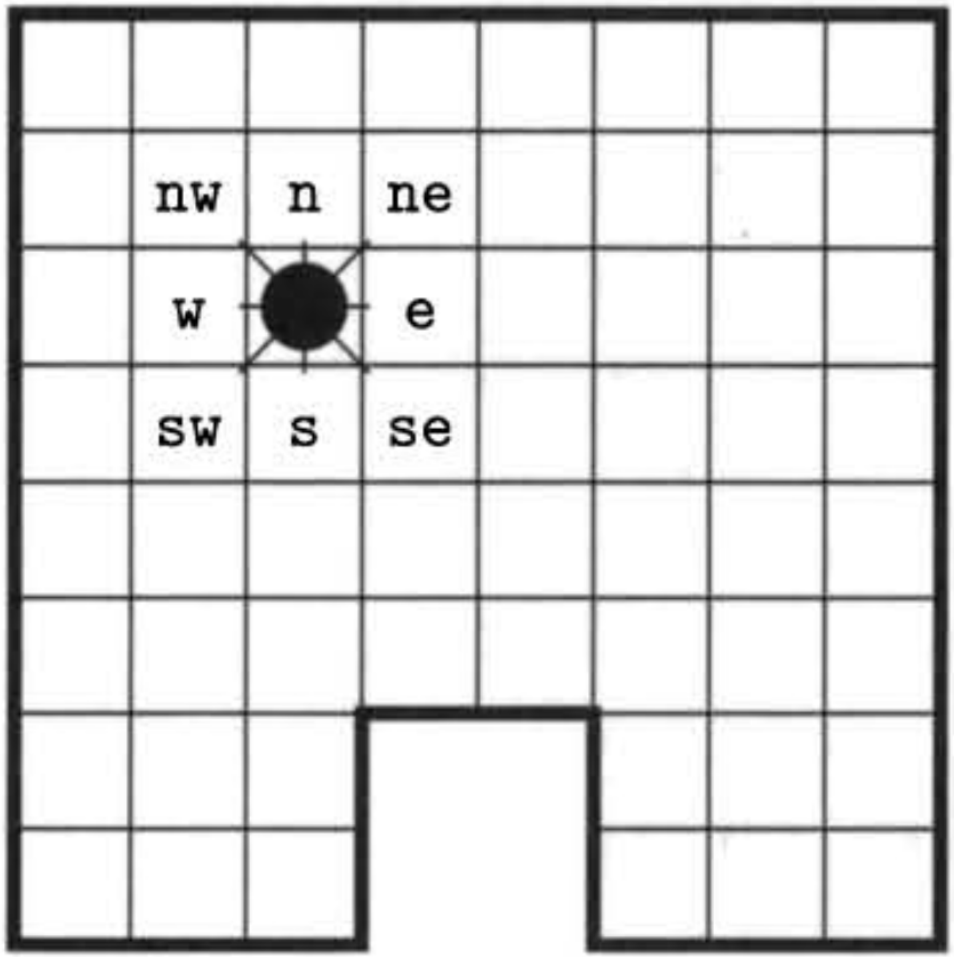
- Minimise $f(x, y, z) = x - xy + 2z, 1 \leqslant x, y, z \leqslant 4$
- Representation: 2 bits for each parameter.
  - $[x, y, z] \rightarrow [XX, YY, ZZ]$
  - $\{1, 2, 3, 4\} \rightarrow \{00, 01, 10, 11\}$
- Example:

Individual	x	y	z	$f(x, y, z)$
011110	2	4	3	0
111110	4	4	3	-6
110100	4	2	1	-2

KING'S College LONDON

Wall-following robot

- Using GP to evolve a wall-following robot.



Wall-following robot

- Build the program up from four primitive functions:
  1. `AND(x, y) = 0 if x = 0; else y`
  2. `OR(x, y) = 1 if x = 1; else y`
  3. `NOT(x) = 0 if x = 1; else 1`
  4. `IF(x, y, z) = y if x = 1; else z`
- And four actions:
  1. North moves one cell up the grid
  2. East moves one cell right in the grid
  3. South moves one cell down the grid
  4. West moves one cell left the grid

KING'S College LONDON

Wall-following robot

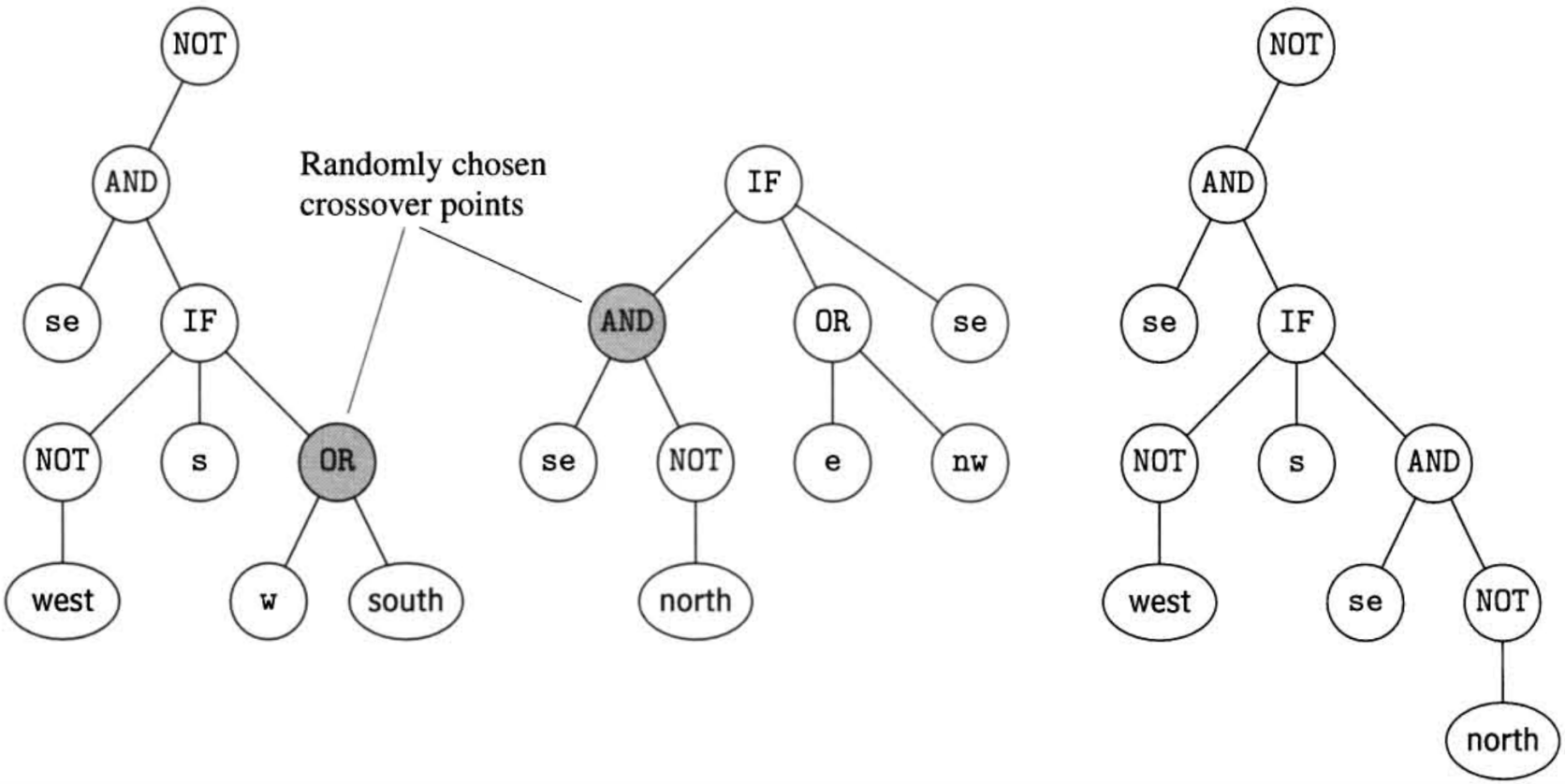
- We must ensure that all expressions and sub-expressions have values for all possible arguments, or terminate the program.
- This makes sure that in any tree constructed (every function is correctly formed) will be an executable program.
- Even if the program is executable, it may not produce “sensible” output.
- For example: it may divide by zero, or generate a negative number where only a positive number makes sense.
- So we always need to have some kind of error handling to deal with the output of individual programs.

KING'S College LONDON

Wall-following robot: Reproduction

- The basic steps:
  - Evaluate the fitness function
  - Select the most fit
  - Breed the most fit
- But how do we breed programs?

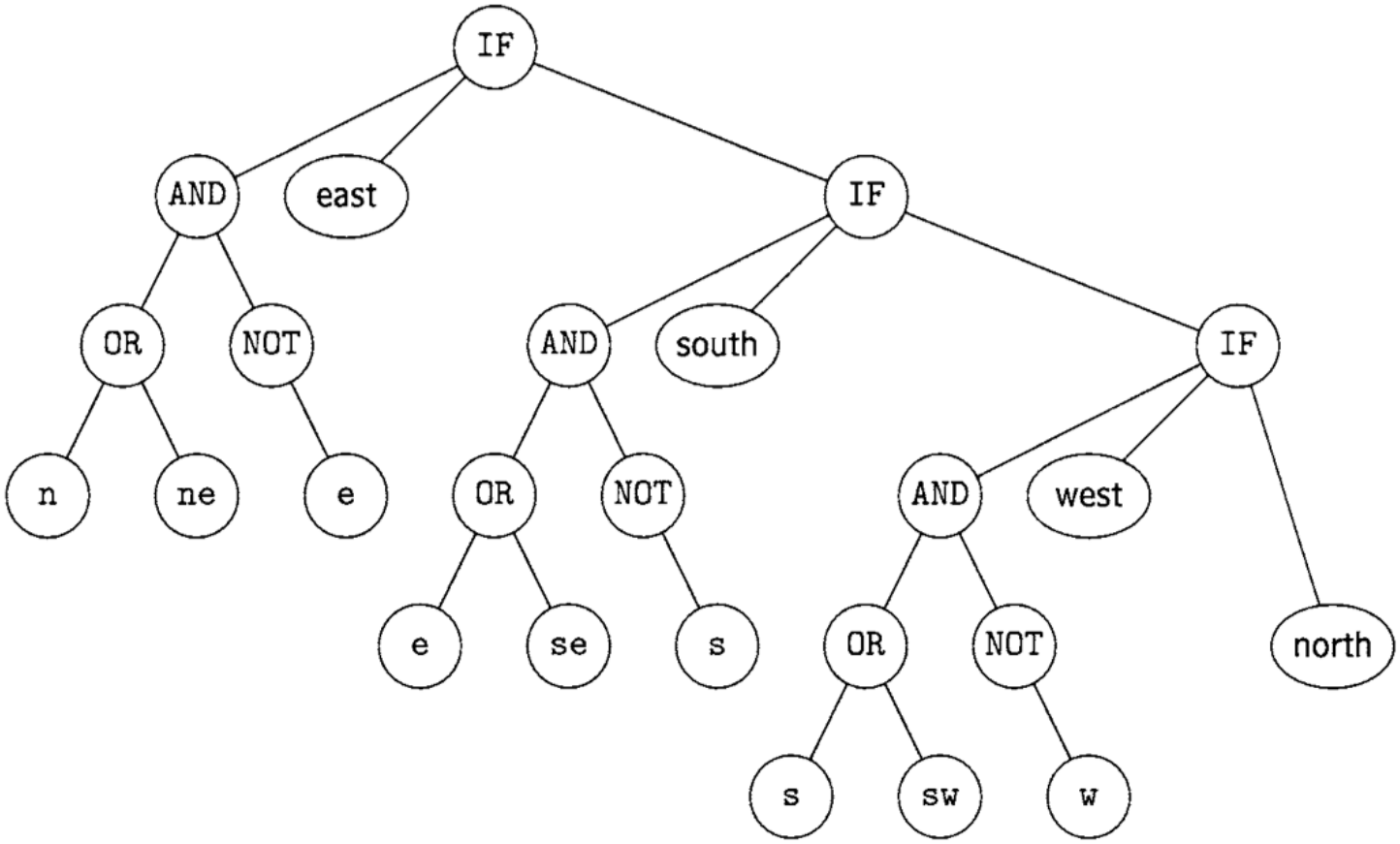
Wall-following robot: Reproduction



Wall-following robot: Before we start

- To give us an idea of what we are looking for, the following slide gives an example program in the GP tree-format.
- This shows that the GP-format is somewhat clumsy.
- However, as we shall see, this program is relatively compact when compared with the programs that will be generated by GP.

Wall-following robot: Example solution




```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
            (west)
            (north))))
```

Wall-following robot: Learning to follow walls

- Start with 5000 random programs.
- Fitness is evaluated by running each program on the task.
- Run the program 60 times and count the number of cells next to the wall visited.
- Do 10 runs from random start points.
- Total count is the fitness.
- Worst possible program gets 0.
- Best possible program gets 32.
  - Over 10 runs, the best fitness is 320.

KING'S College LONDON

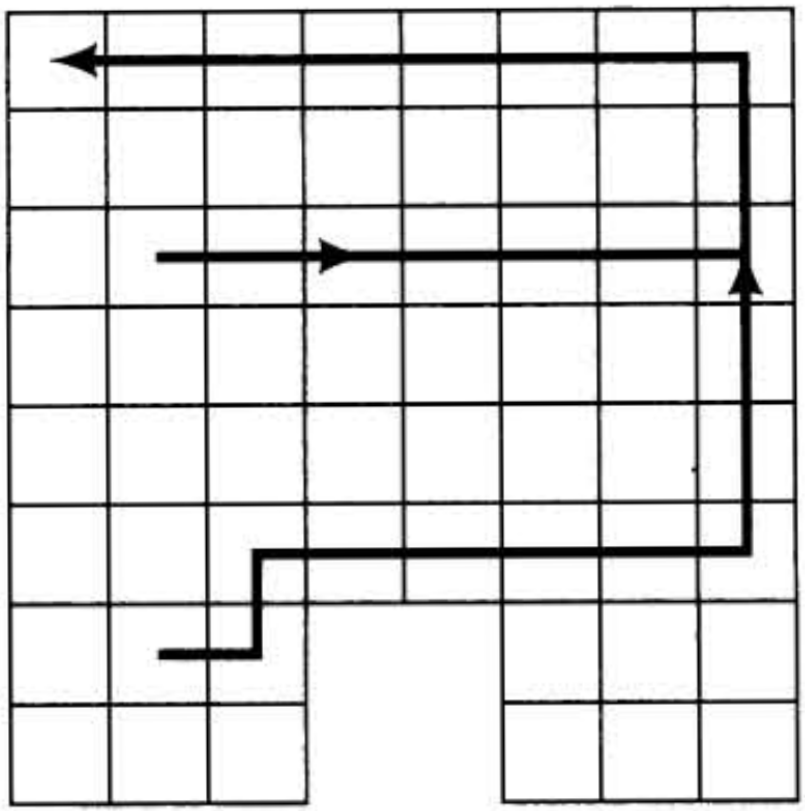
Wall-following robot: Learning to follow walls

- Take 500 programs and add them to the next generation.
- Choose them by tournament selection:
  - Pick 7 programs at random.
  - Add the most fit to the next generation.
- Then create 4500 children into the next generation where parents are chosen by tournament selection and a crossover operation is applied to parents.
- Mutate by replacing a randomly chosen subtree with a randomly generated subtree.

KING'S College LONDON

Wall-following robot: Generation 0

- The most fit member of the randomly generated initial programs has a fitness of 92, and has the kind of behaviour shown below.
- The program itself is given in the next slide.

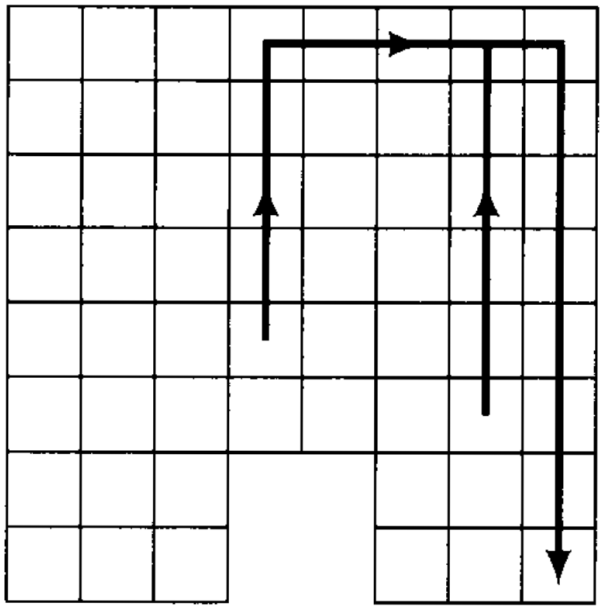


Wall-following robot: Generation 0

```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
                   (NOT (NOT (north))))))
     (IF (OR (NOT (AND (IF (sw)(north)(ne))
                       (AND (south)(1))))
             (OR (OR (NOT (s))
                     (OR (e)(e)))
                 (AND (IF (west)(ne)(se))
                      (IF (1) (e)(e)))))
         (OR (NOT (AND (NOT (ne))(IF (east)(s)(n))))
             (OR (NOT (IF (nw)(east)(s)))
                 (AND (IF (w)(sw)(1))
                      (OR (sw)(nw)))))
         (OR (NOT (IF (OR (n)(w))
                      (OR (0)(se))
                      (OR (1)(east))))
             (OR (AND (OR (1)(ne))
                      (AND (nw)(east)))
                 (IF (NOT (west))
                     (AND (west)(east))
                     (IF (1)(north)(w)))))))
```

Wall-following robot: Generation 2

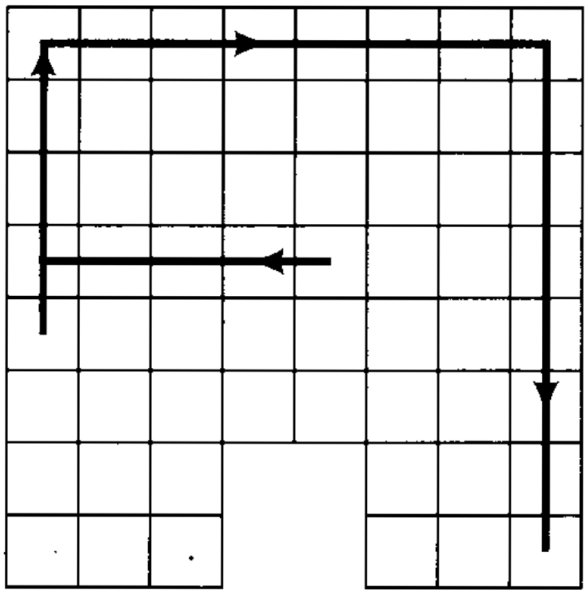
- The most fit member of generation 2 has fitness 117.



```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```

Wall-following robot: Generation 6

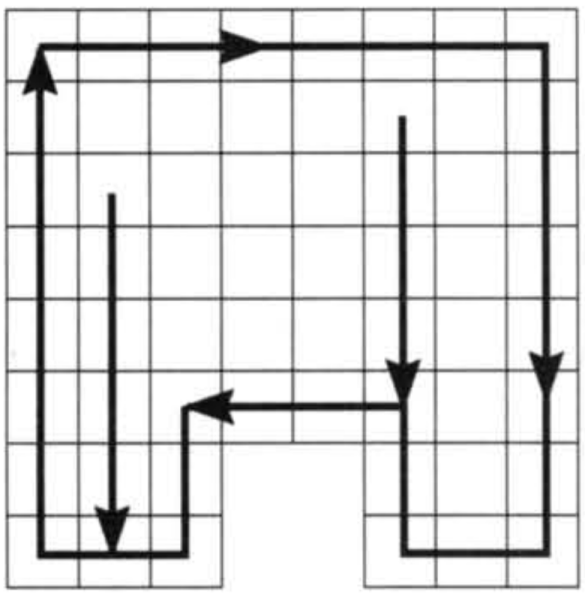
- The most fit member of generation 6 has fitness 163.
- The program follows the wall perfectly, but gets stuck in the lower-right corner.



```
(IF (AND (NOT (e))
         (IF (e)(s)(nw)))
    (OR (IF (1)(e)(south))
        (IF (north)(east)(nw)))
    (IF (OR (AND (0)(north))
            (AND (e)(IF (e)
                        (IF (se)(south)(east))
                        (north))))
        (AND (e)
             (NOT (IF (s)(sw)(e))))
        (OR (OR (AND (nw)(east))
                (west))
            (nw))))
```

Wall-following robot: Generation 10

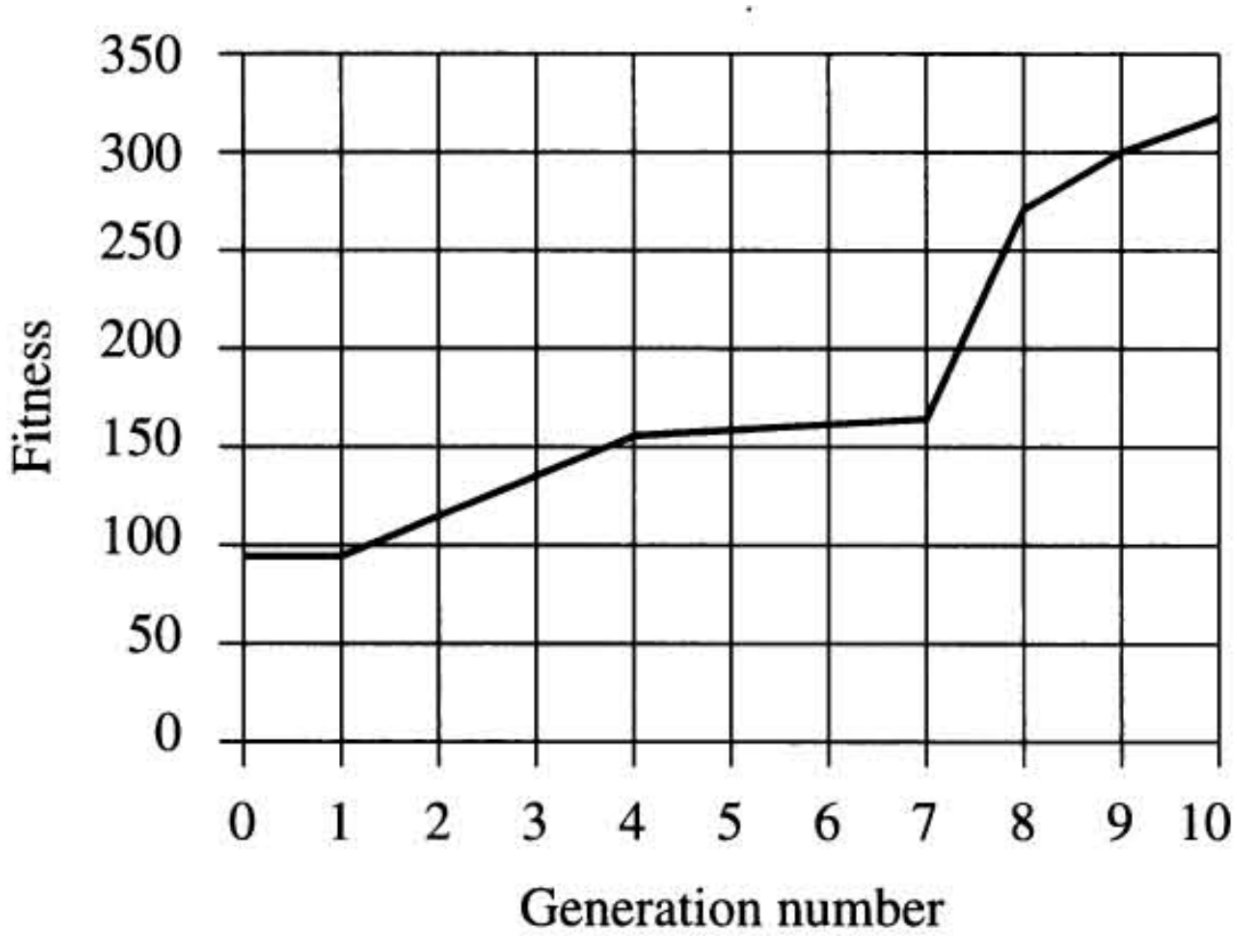
- The most fit member of generation 10 has fitness close to the maximum 320.
- The program follows the wall perfectly, heading south until it reaches the boundary.



```
(IF (IF (IF (se)(0)(ne))
        (OR (se)(east))
        (IF (OR (AND (e)(0))
                (sw))
            (OR (sw)(0))
            (AND (NOT (NOT (AND (s)(se))))
                 (se))))
    (IF (w)
        (OR (north)
            (NOT (NOT (s))))
        (west))
    (NOT (NOT (NOT (AND (IF (NOT (south))
                            (se)
                            (w))
                        (NOT (n)))))))
```

Wall-following robot: Results of learning

- This graph shows how the fitness of individuals grows quite sharply over the ten generations.



References

Mitchell Tom M. Mitchell (1997), Machine Learning, McGraw-Hill. chapter 9

WFH Ian H. Witten, Eibe Frank and Mark A. Hall (2011), Data Mining: Practical Machine Learning Tools and Techniques (3rd ed.), Morgan Kaufmann.

Nilsson Nils J. Nilsson (1998), Artificial Intelligence: A New Synthesis, Morgan Kaufmann.

KING'S College LONDON

Some classic literature on evolutionary algorithms

- John H. Holland (1962), Outline for a logical theory of adaptive systems, *Journal of the Association for Computing Machinery*, 3, pp.297-314.
- John H. Holland (1975), *Adaptation in natural and artificial systems*, University of Michigan Press.
- John Koza (1992), *Genetic programming: On the programming of computers by means of natural selection*, MIT Press.
- Peter J. Angeline and Jordan B. Pollack (1993), Competitive Environments Evolve Better Solutions for Complex Tasks, *Genetic Algorithms: Proceedings of the Fifth International Conference*.
- Jordan B. Pollack and Alan D. Blair (1998), Co-Evolution in the successful learning of backgammon strategy, *Machine Learning*, 32, pp.225-240.

KING'S College LONDON

Summary

- Genetic Algorithms
- Genetic Programming
- Co-evolutionary Algorithms
- Examples