Bio-Inspired Algorithms

Evolutionary Algorithms

Daniel Vaz
daniel.ramosvaz@esiee.fr

2026-04-02

Recap

Evolutionary Algorithms

- Survival of the fittest
  - Positive characteristics pass on to the next generation
- Population-based meta-heuristic
- Adaptive, creative and transparent

Mechanisms

- Variation: mutation and recombination
- Parent and survivor selection

Applications

- TSP, Eight-Queens, Knapsack, Structure Approximation

Approaches

Last week

- Genetic algorithms
  - Use evolution for combinatorial optimization
- Evolution strategies
  - Continuous optimization, with self-adjusting parameters
- Genetic programming
  - To design automata or programs

Approaches

Today

- Genetic algorithms
  - Use evolution for combinatorial optimization
- Evolution strategies
  - Continuous optimization, with self-adjusting parameters
- Genetic programming
  - To design automata or programs

Genetic Algorithms

- Encode individuals using a genotype
- Evolve population over generations
  - Using recombination and mutation, then selection

Algorithm

- Start with a random population
- Every iteration:
  - Recombine some pairs from the population
  - Apply (rare) mutations to the resulting solutions
  - Select the next population

Genetic Algorithms

Algorithm GeneticAlgorithm

1: $P :=$ InitializePopulation
2: Evaluate(P)
3: **while** not Finished **do**
4: $\quad M :=$ SelectParentPairs(P)
5: $\quad C :=$ Recombine(M)
6: $\quad$ Mutate(C)
7: $\quad$ Evaluate(C)
8: $\quad P :=$ SelectNext(P, C)
9: **end while**

Encoding

Encoding: Link between genotype and phenotype

- Depends on the problem
- Also on the information we want to change

Genetic algorithms: discrete representations

- Binary strings, sequence of values, ...
- Ensure that it stays valid (permutation)

General principles:

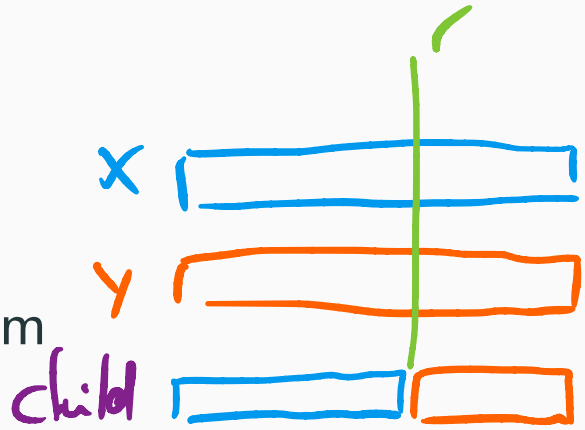
- Simple structure, such as binary
  - Makes variation operators easier
- Might not be space-optimal

Variation Operators

Operators : How to generate good new individuals

Recombination

- Operator that combines two individuals
- Usually generates one or two new ones
- Intends to aggregate good features
- **Single crossover:** $X[:r] + Y[r:]$, r random



Mutation

- Modifies an individual
- Small variations to introduce unique features
- **Single bit:** flip bit i for random i

Selection methods

Selection: serve to improve population

- Favors good individuals in recombination and survival

Parent selection

- Which individuals participate in recombination
- Usually random, but using fitness as guide
- **2-tournament:** pick two random individuals, keep best
    - Similar for the second parent

Survivor selection

- Selects which individuals pass to the next generation
- Among previous generation and children
- Uses fitness more directly
- **Best overall:** take the best n out of population+children

Selection

Selection: serve to improve population

- Favors good individuals in recombination and survival
- Maintains genetic diversity

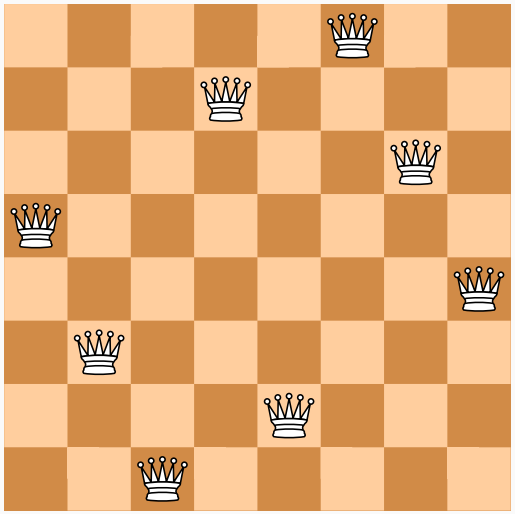
Important properties

- Selection pressure: how much do we favor the best individuals
  - Influences how quickly we converge
  - Fast convergence is not always good
- Genetic drift: loss of variety over time
  - Influences the quality of solutions
  - Sometimes the best solutions can disappear

Application – The Eight-Queens Problem

Problem:

- Place n queens in an $n \times n$ chessboard
- One queen per row, column, diagonal



Algorithm

- Representation: permutation
- Recombination: Permutation insertion
- Mutation: Swap (prob. 80%)
- Parents: 2 out of random 5
- Survivor: Replace worst

Genetic Programming

Genetic Programming

Evolutionary algorithms for designing programs

Individuals

- Programs or sequences of instructions
- Good representation is important

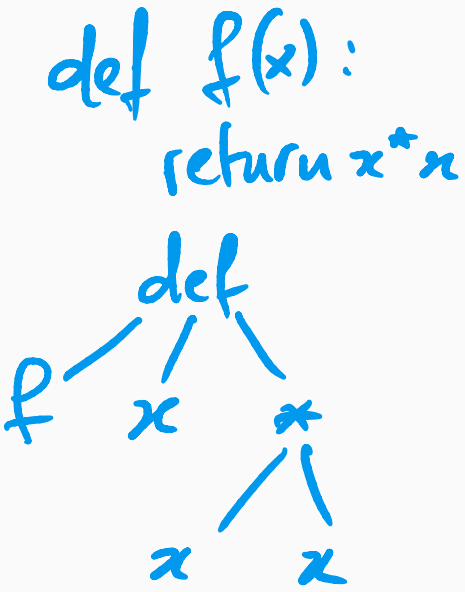
Fitness

- How close to intended result
- Usually evaluated on test data (input)
    - Run program and compare with ground truth

Encoding

Representation

- Genotype is the code
- Phenotype is the program and its results
- Genotype and phenotype are very different
- Two common genotypes: trees and automata



Trees

- Capture expressions in a formal syntax
- E.g. arithmetic expressions, logic formulas, code

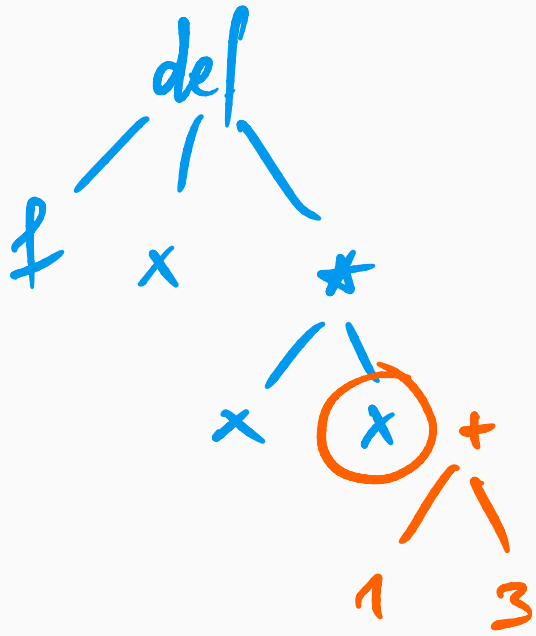
Automata

- Set of states and transition rules

Variation Operators

Mutation

- Replace a node by random construction
  - Trees: replace by a new random subtree
- Usually quite low probability (0-5% overall)
  - Contrast with bit flip: mutation w.p. 63%

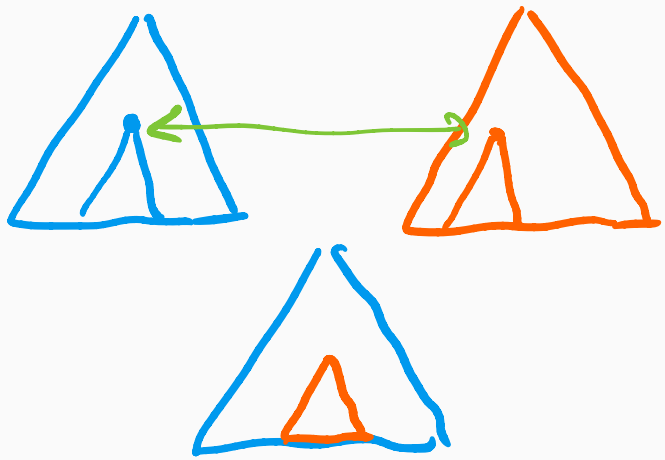


Variation Operators

Mutation

- Replace a node by random construction
  - Trees: replace by a new random subtree
- Usually quite low probability (0-5% overall)
  - Contrast with bit flip: mutation w.p. 63%

Recombination

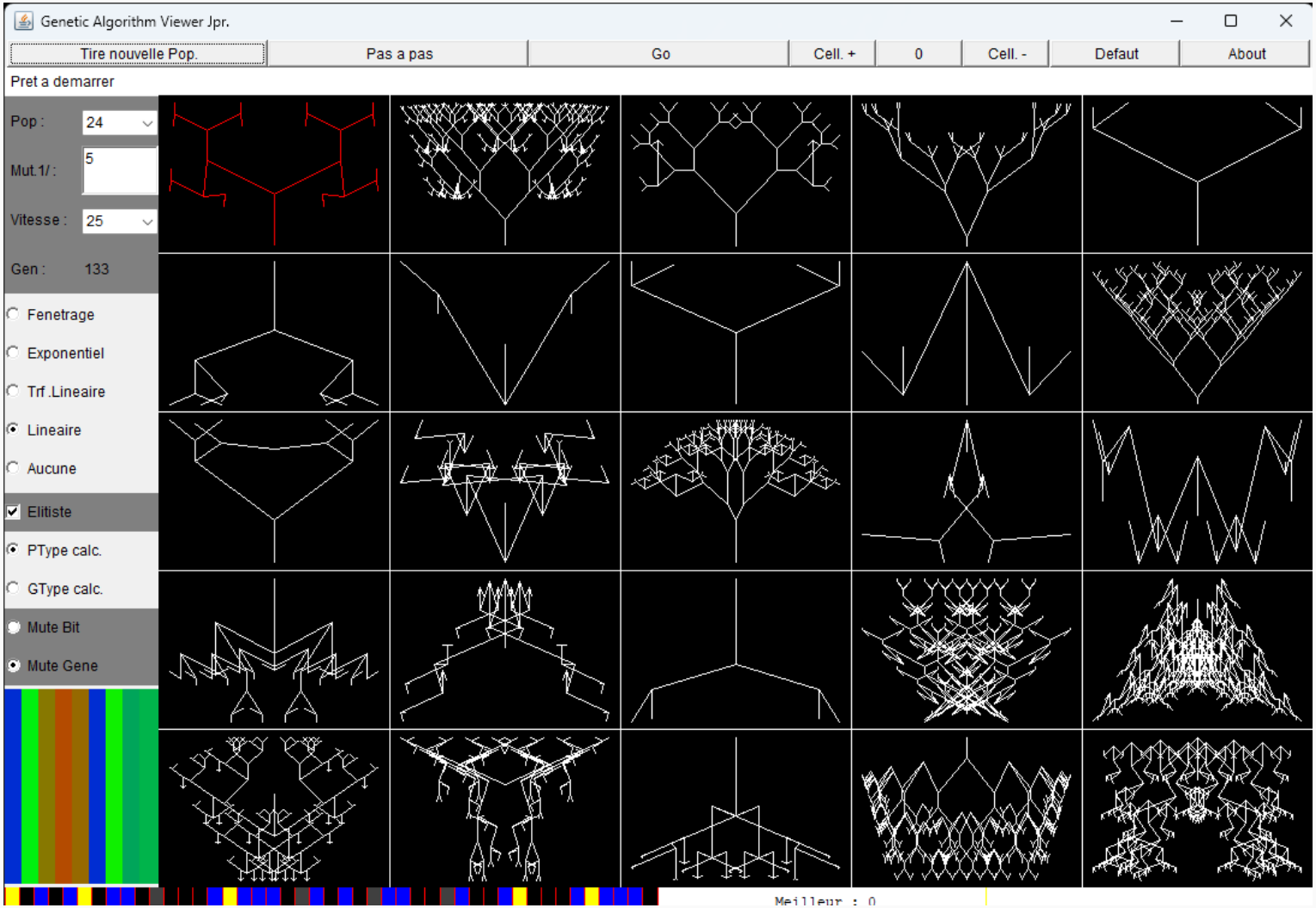


- Exchange two nodes completely
  - Trees: exchange subtrees
- Depth can increase
- To improve fitness, must exchange similar nodes

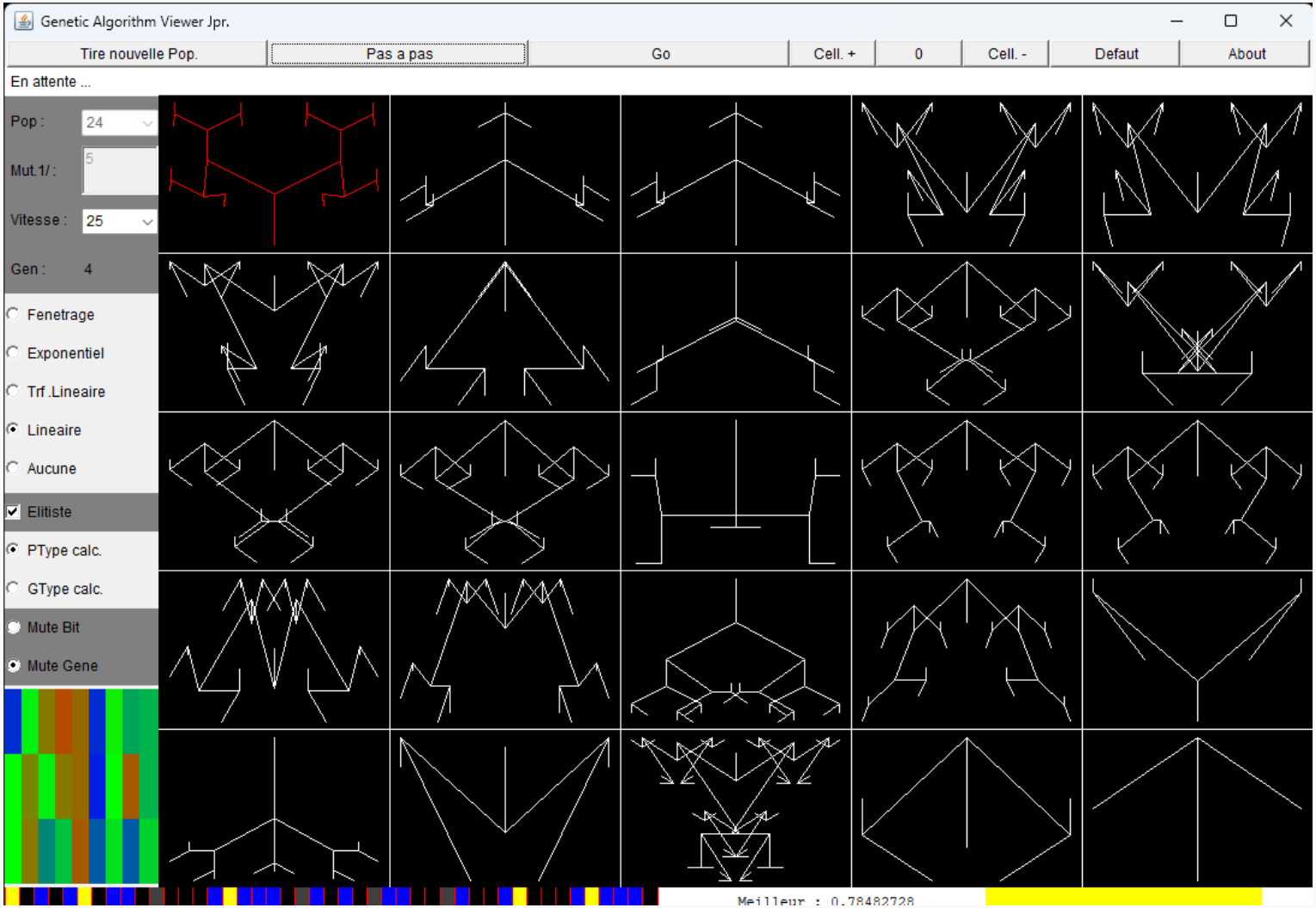
Genetic Programming

Applications

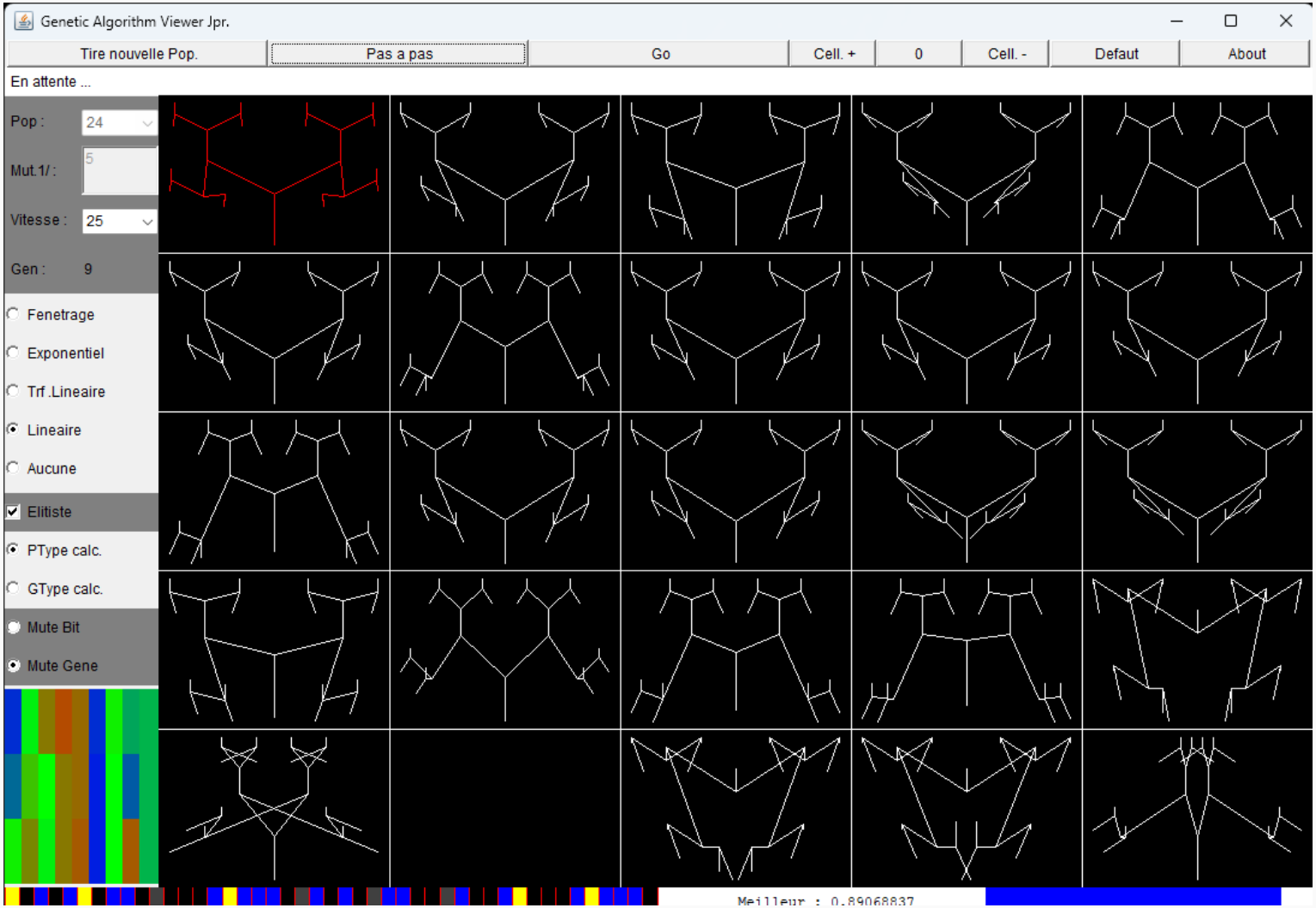
Tree shape approximation



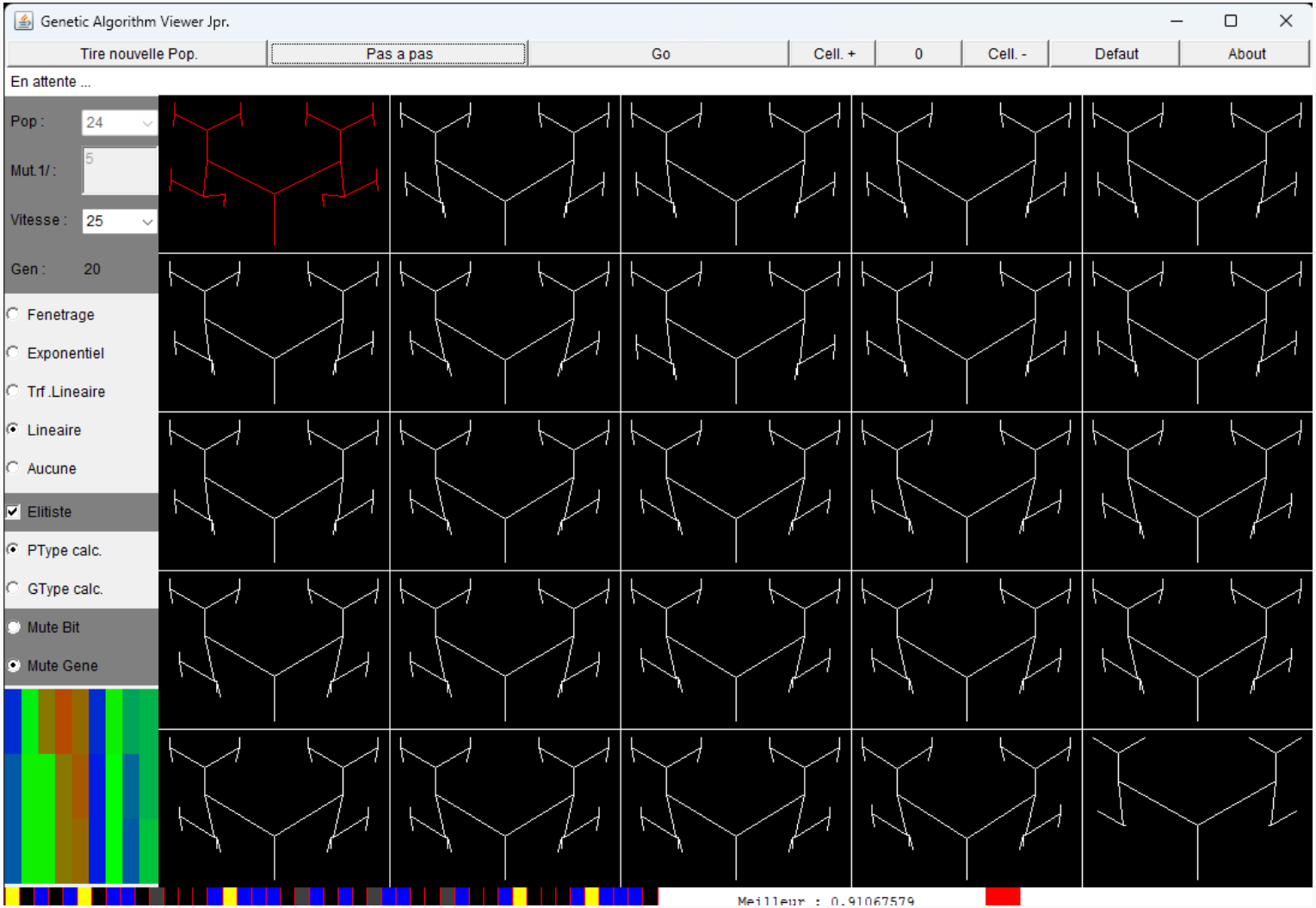
Tree shape approximation



Tree shape approximation



Tree shape approximation



Tree shape approximation

Recursive drawing from branching parameters

Genotype

- Number of iterations
- Length decay
- Branch directions

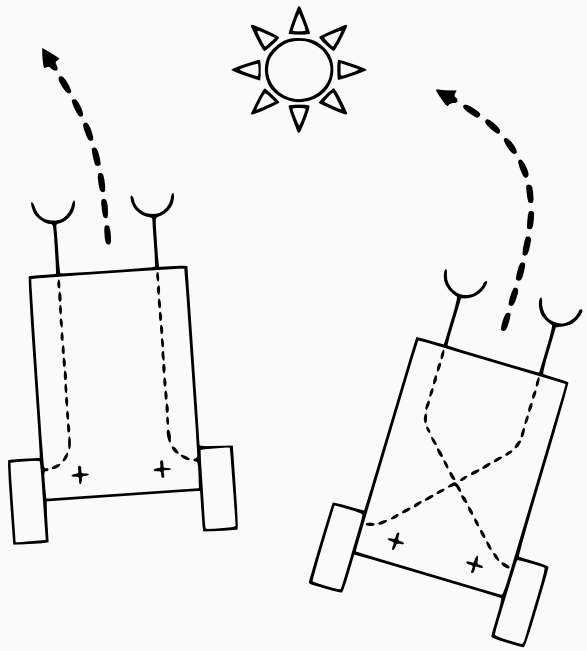
Phenotype:

- Drawing of the tree

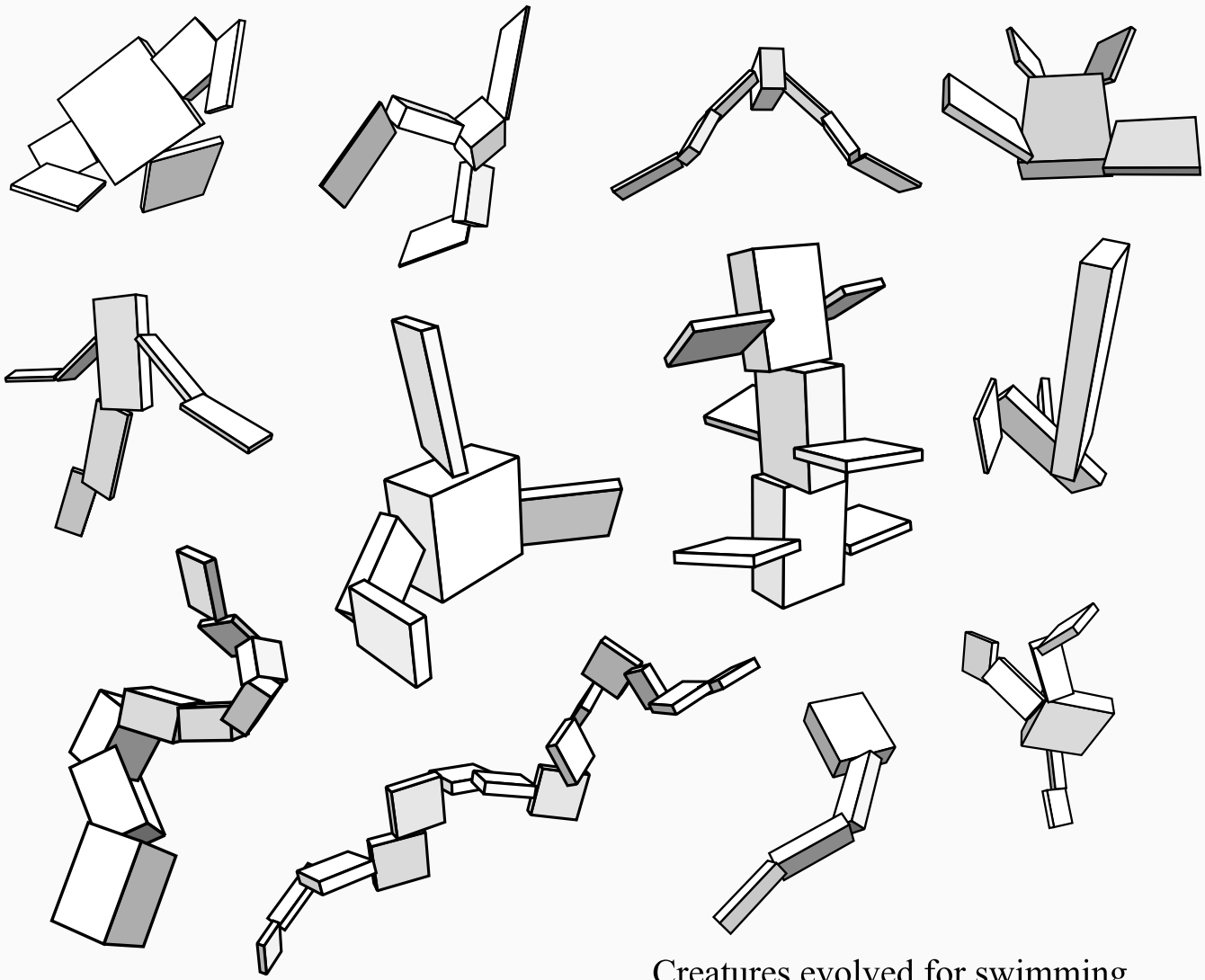
Braitenberg Machines

Problem

- Train robot to follow light source
- Based on simple neural networks
- Hybrid model
  - Genotype is numeric (matrix)
  - Fitness is programmatic (execution)



Robot Locomotion



Creatures evolved for swimming.
Karl Sims. *Evolving Virtual Creatures*

Robot Locomotion

Problem: Virtual 3D creatures learning complex movement

- Creatures composed of blocks with articulation
- Learned movement: jump, swim, fly

Genotype

- Block description
  - Position and size
  - Or stored as generator program
- Learned movement
  - As sequences of simple moves or program

Phenotype

- Physical appearance
- The appearance and result of the movement

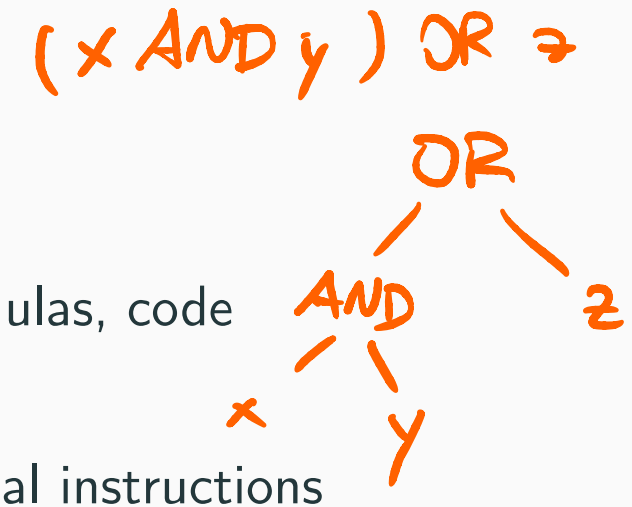
Genetic Programming

Encoding

Encoding

Tree representation

- Capture expressions in a formal syntax
  - E.g. arithmetic expressions, logic formulas, code
- Genome encodes a tree structure
  - Leaves correspond to values or terminal instructions
  - Internal nodes operate on their children



Encoding

Tree representation

- Capture expressions in a formal syntax
  - E.g. arithmetic expressions, logic formulas, code
- Genome encodes a tree structure
  - Leaves correspond to values or terminal instructions
  - Internal nodes operate on their children

Automaton representation

- Based on the concept of states
  - Each state dictates a different behavior
  - Automaton reacts to current position of the input
  - Can move to a different state and/or position
- Connected to Turing machines

Encoding

Learning classifier systems

- Solutions are sets of `if-then` rules
- Encoding as a sequence of parameters for each rule

Other representations

- Sequences of instructions
  - Linear genetic programming and Multi expression programming
  - Uses simple instructions, such as `R4 = R2 + R3`
  - By encoding in bytecode, allows bit mutations
- Directed multigraphs
  - Allows looping and reuse
- Neural networks

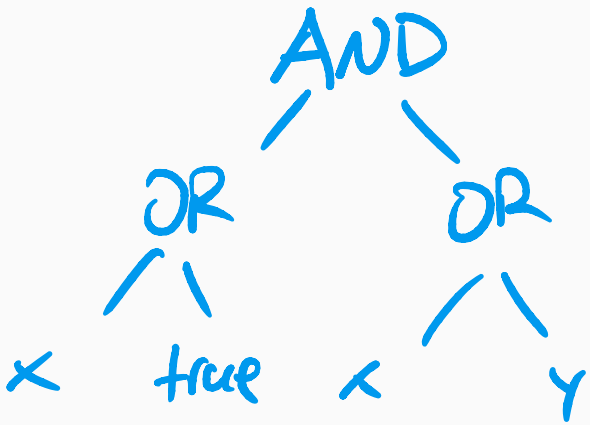
Encoding

Representation

- Genotype: code; Phenotype: program
- Genotype and phenotype are very different
- Usually there is a maximum depth / size

Introns: pieces of non-operating code

- Useful for variation operators
- Lead to better performance



Genetic Programming

Tree Representation

Tree Representation

Idea: Represent a program as a tree of operations

Types:

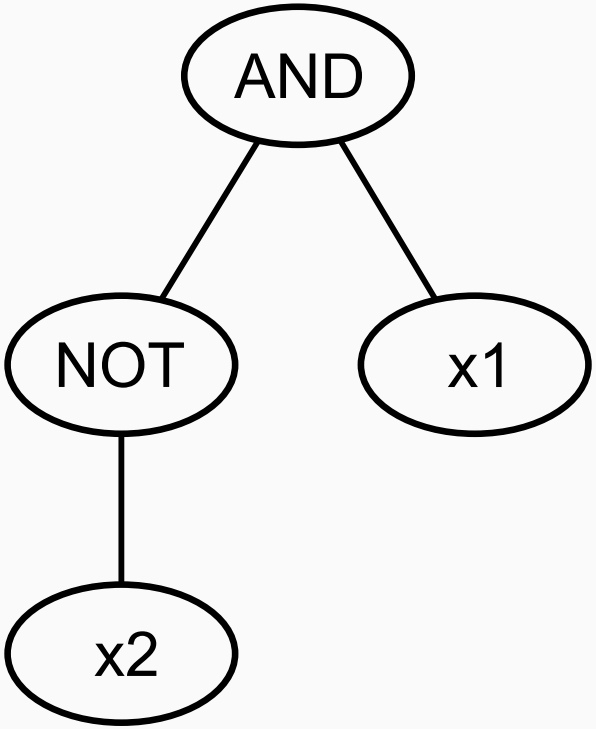
- Functional: nodes are operators or values
  - Represent functions, operate on values
- Imperative: nodes are instructions
  - Represent programs, operate on states

Nodes: can be internal or terminal

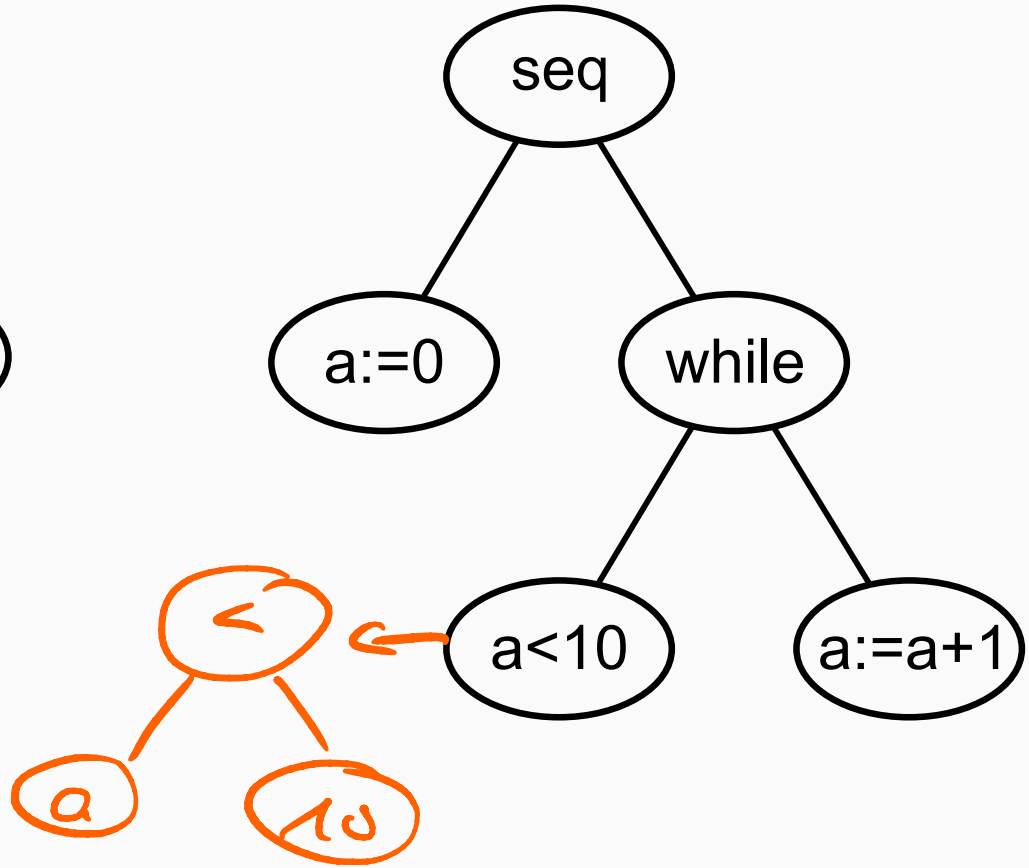
- Internal nodes are operations with arguments
- Terminal nodes are values or independent operations

Tree Representation

Functional



Imperative



Functional Tree Encoding

Terminal nodes

- Constants, variables
- Functions without arguments
    - E.g. Random number generation

Internal nodes : binary operations

Imperative Tree Encoding

Terminal nodes: independent instructions

- Assignments
- IO operations: print, reading a file

Internal nodes: flow control

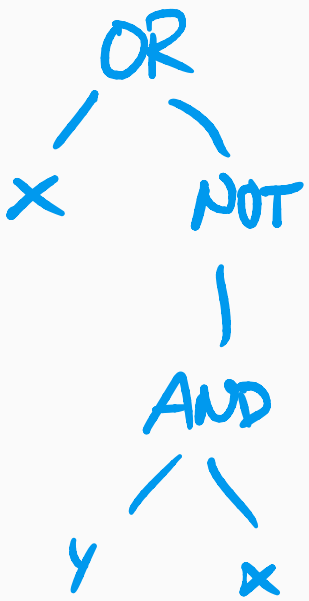
- `seq`: execute `child1` then `child2`
- `if-then`: if `child1` is true, execute `child2`
- `while-do`: while `child1` is true, execute `child2`

Tree Initialization

AND, OR, NOT, x, y

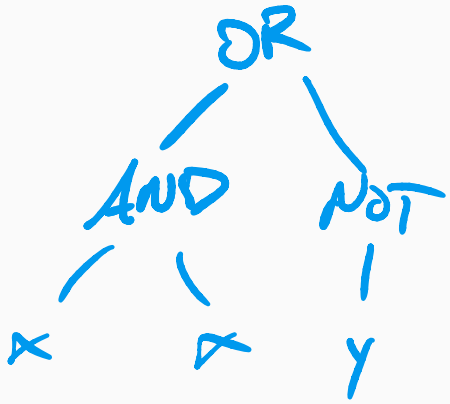
Grow: (random individual)

- Chooses a random node (starting from the root)
- If the node has children, generate children recursively
- Usually has a depth limit



Full (random individual)

- Like grow, but every branch has full depth



Tree Initialization

Grow: (random individual)

- Chooses a random node (starting from the root)
- If the node has children, generate children recursively
- Usually has a depth limit

Full (random individual)

- Like grow, but every branch has full depth

depth = 5

G2 G3 G4 G5

F2 F3 F4 F5

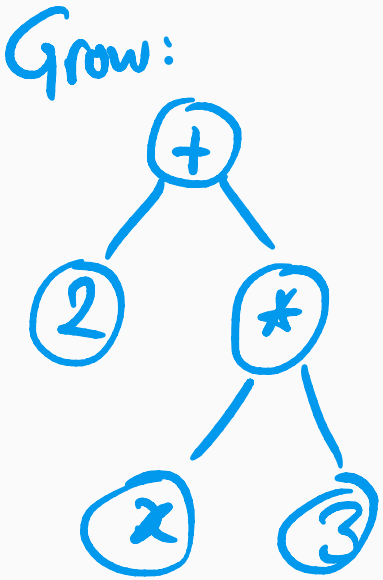
Ramped half-and-half (random population)

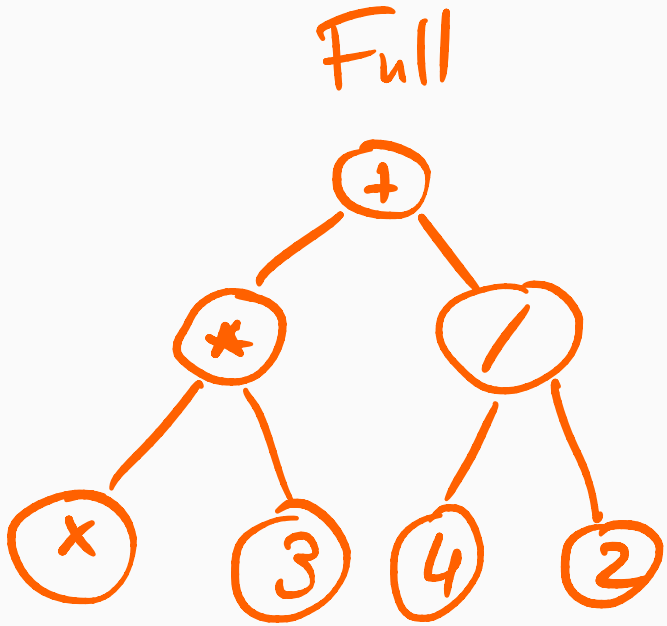
- Creates individuals of every possible depth
- Half of the individuals generated using grow, half using full

Tree Initialization

Exercise: Construct trees of max. depth 3 using the grow and full

- Random sequence: $+, 2, *, x, -, 3, 2, /, 4, *, 2, y, 3, /, +, 2, 1$





Tree Initialization

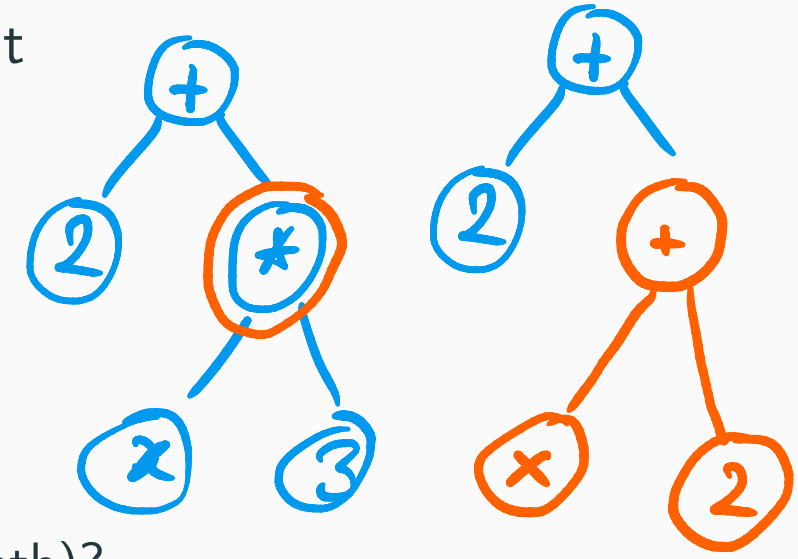
Exercise: Construct trees of max. depth 3 using the grow and full

- Random sequence: $+, 2, *, x, -, 3, 2, /, 4, *, 2, y, 3, /, +, 2, 1$

Tree Operators

Mutation: Random subtree replacement

- Select a node at random
- Replace it with a random subtree
  - Using initialization procedures
  - Grow or full?
- Depth:
  - Use max. depth (minus node depth)?
  - Use depth of existing subtree?



Tree Operators

Mutation: Random subtree replacement

- Select a node at random
- Replace it with a random subtree
  - Using initialization procedures
  - Grow or full?
- Depth:
  - Use max. depth (minus node depth)?
  - Use depth of existing subtree?

Subtree crossover

- Select at random one node of each parent
- Exchange the subtrees at those nodes
- Might exceed maximum depth

Genetic Programming

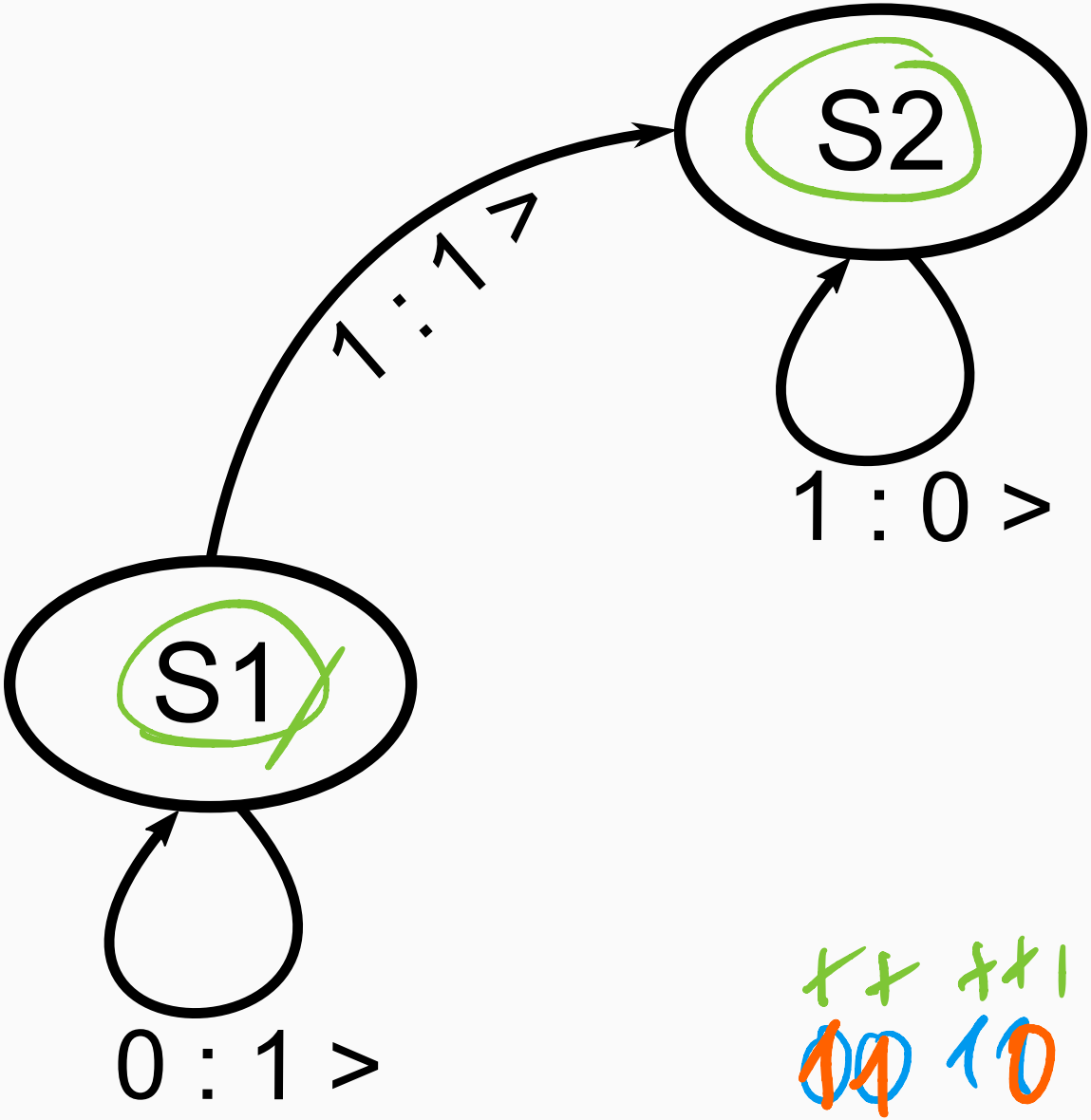
Automata Representation

Automata

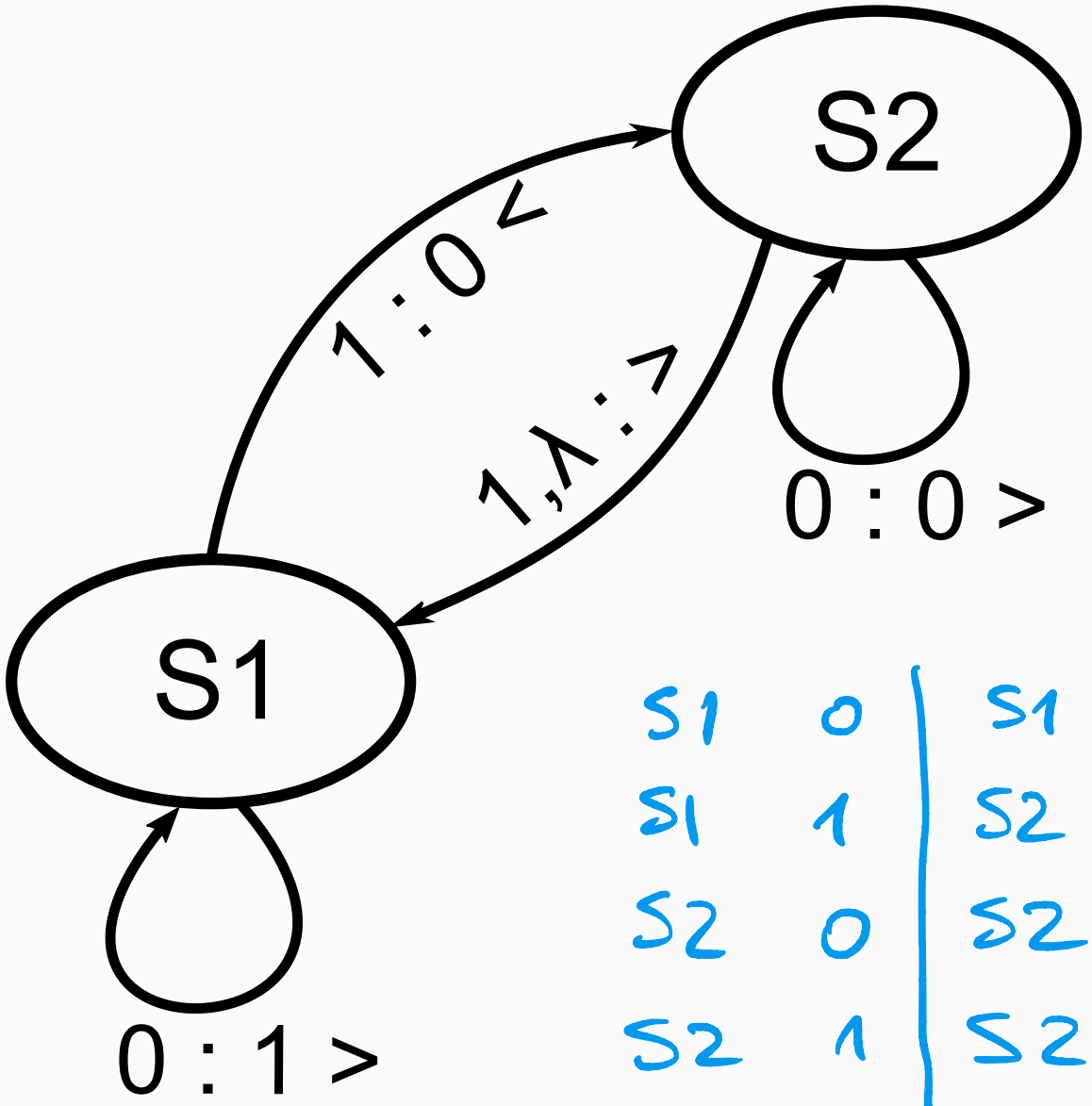
Automaton

- State: current node and memory position
  - Memory starts with input
- Action depending on node and memory value:
  - Transition: which node to go to
  - Write value: value to write to memory
  - Change to position: change memory position by ± 1
- Connected to Turing machines

Automata



Automata



S1	0	S1	1	>
S1	1	S2	0	<
S2	0	S2	0	>
S2	1	S2	λ	>

Encoding

Vector encoding

- Action for every combination of state and read value
- For each action, store transition, write value, position change
    - As three different vectors

Variation operators

- Can use same operators as for vectors
    - Use same position for all vectors (mutation / recombination by columns)

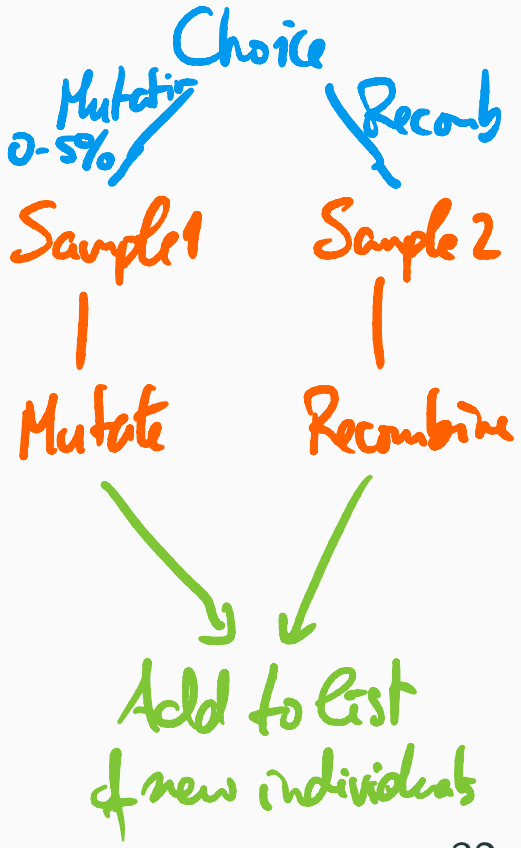
Genetic Programming

Characteristics

Use of Variation Operators

Variation operators: Usually, only one at a time

- Either mutation or recombination
- New population constructed iteratively:
  - Select mutation or recombination at random
  - Select 1-2 random parents for the operator
  - Apply the operator to obtain 1-2 children



Use of Variation Operators

Variation operators: Usually, only one at a time

- Either mutation or recombination
- New population constructed iteratively:
  - Select mutation or recombination at random
  - Select 1-2 random parents for the operator
  - Apply the operator to obtain 1-2 children

Probability of mutation

- Usually quite low probability (0-5% overall)
  - Counteracts high impact of mutation
- Recombination has a large shuffling effect

Over-Selection

Over-selection

- Method used to deal with large populations
  - Population sizes in the thousands are not unusual
- Rank population and divide into top (x%) and bottom
  - x chosen so the top group has size around 100-200
- Parent selection is skewed towards top group (80%)
- An alternative can be to use k-tournaments (high k)
  $k = 10$

Bloat

- When individuals are too large
  - Such individuals tend to appear
  - And then dominate the population
- Reasons are not fully known
  - But large individuals are more complex
  - And so might overfit the data
- Counter-measures:
  - Forbid operations exceeding a depth limit
  - Penalty term in the fitness formula

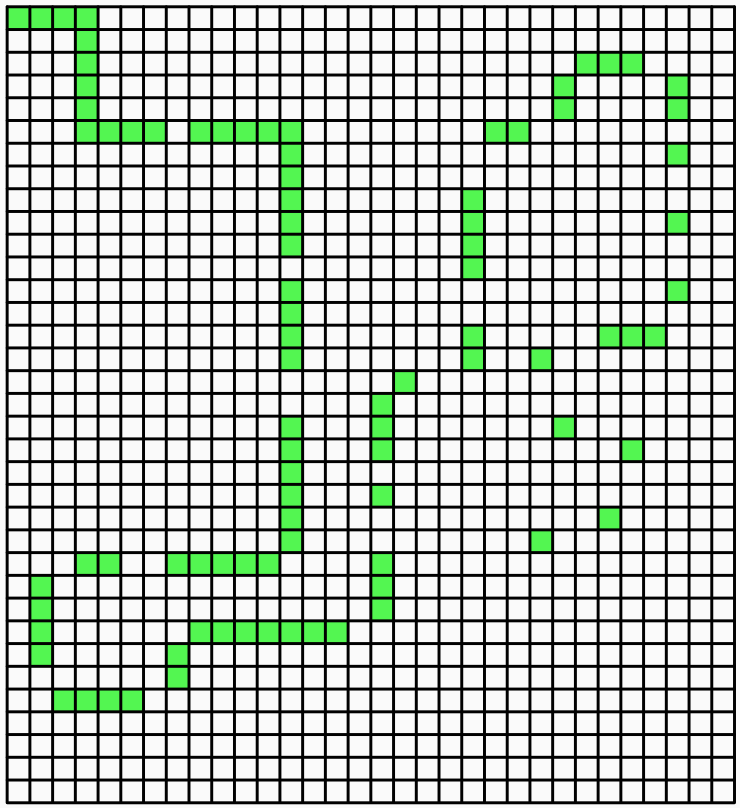
Genetic Programming

Application

Santa Fé Trail

Problem

- Find food in a map
  - As much as possible
- Starting top-left
- Actions:
  - Eat (automatic)
  - Advance
  - Turn left / right
- Perception:
  - Food in the cell one ahead



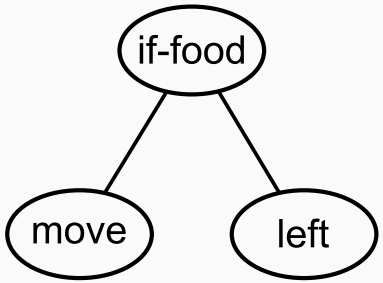
Encoding

Use a tree representation

- Terminal nodes: actions
  - advance, left, right
- Internal nodes:
  - `seq`: Sequence node
  - `if-food`: if it sees food, execute child 1, else child 2

Example: `if-food(move, left)`

- if it sees food, advance,
- else, turn left



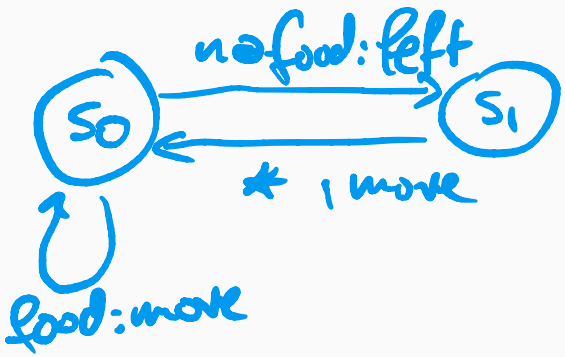
Characteristics

Simple behavior

- Few possible operations
- No memory

Automata can be a better fit

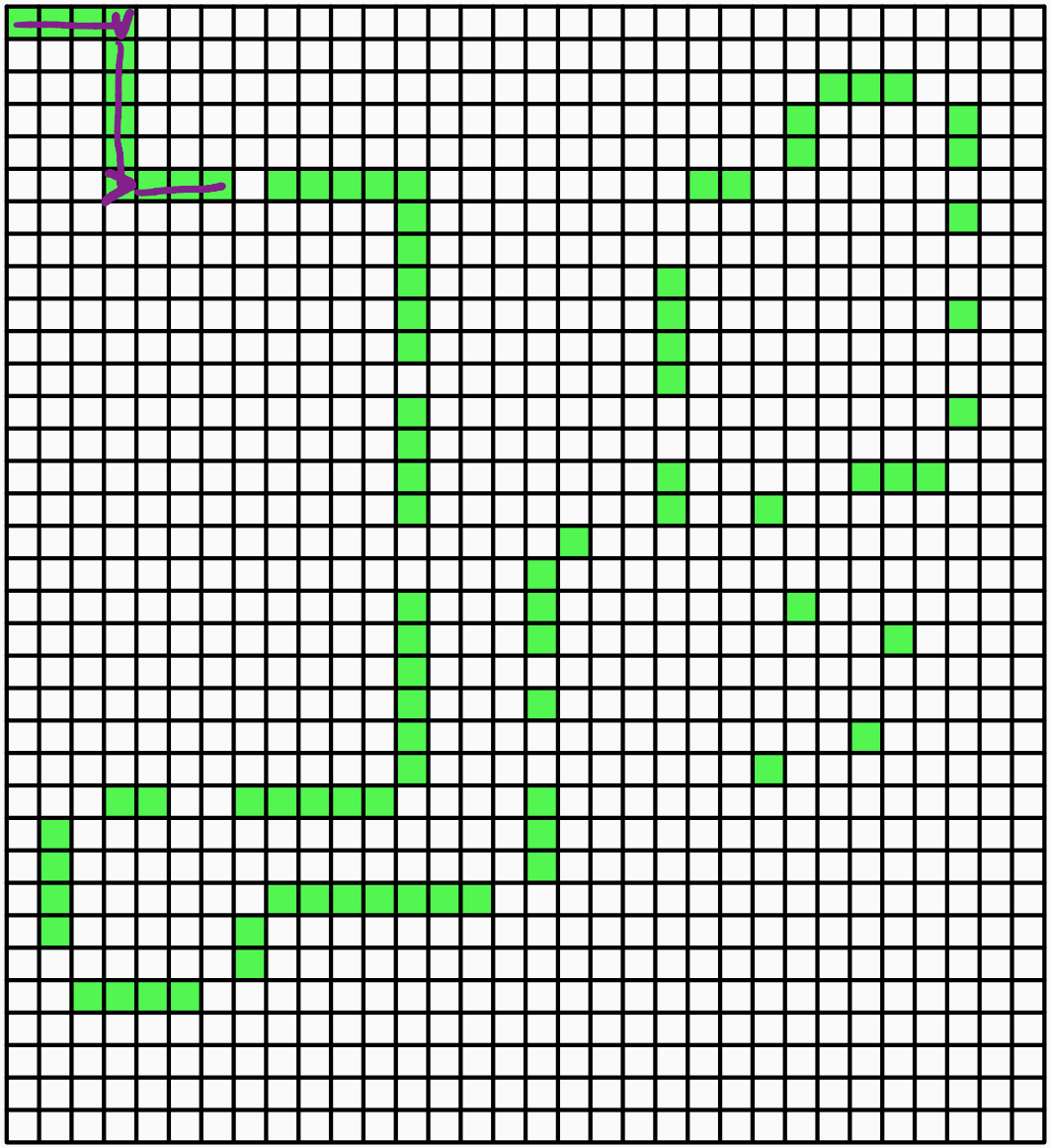
- States allow for limited memory

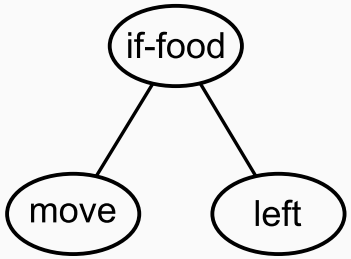


Algorithm evolves the trees / automata

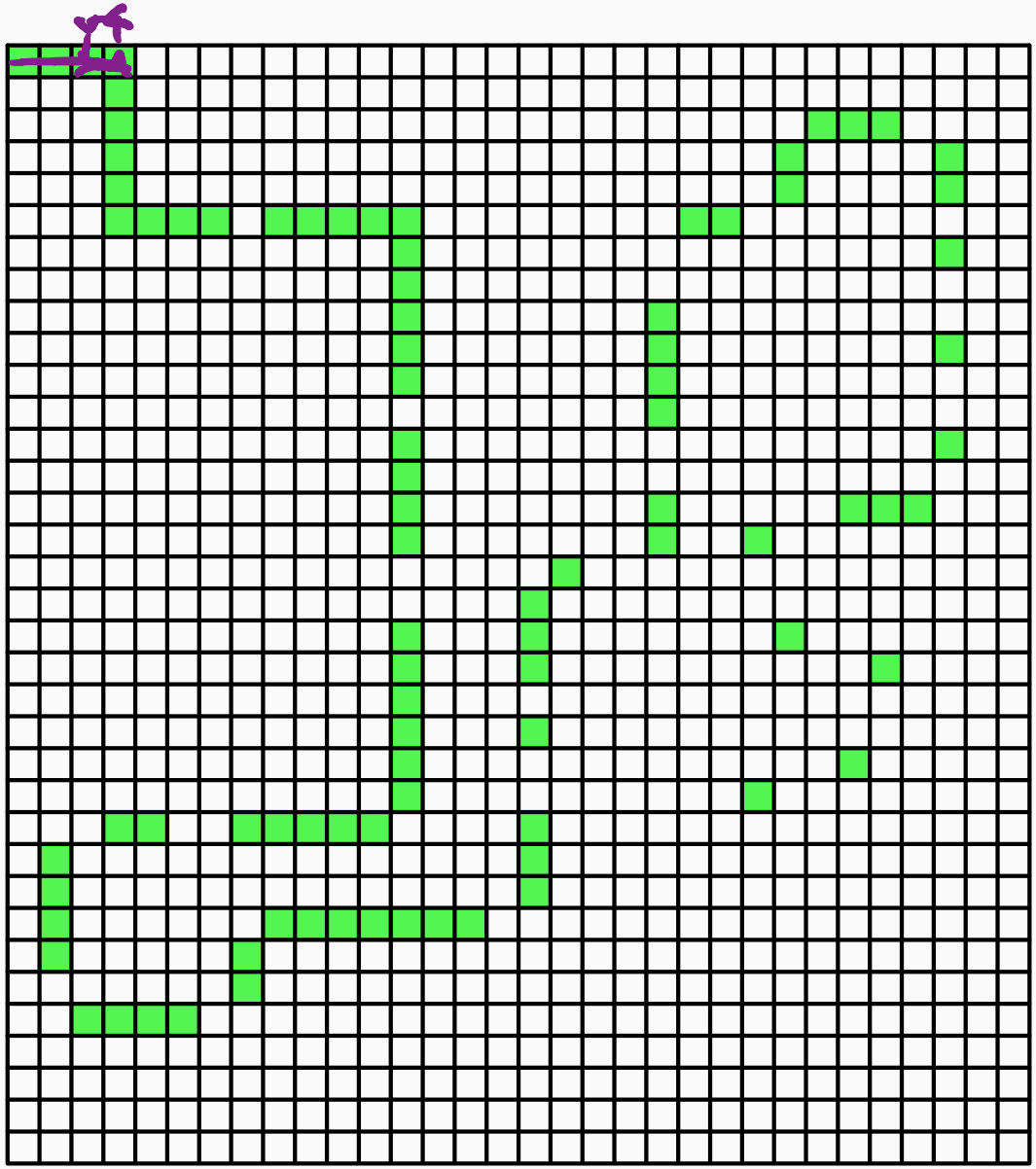
- By applying variation and selection
- To obtain new generations

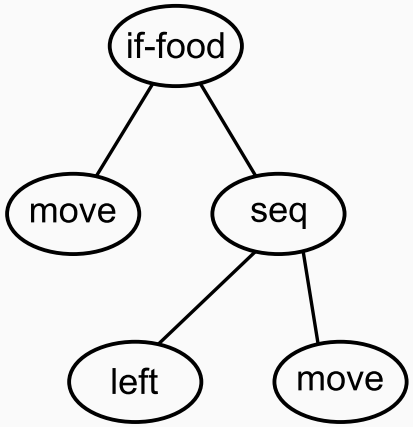
Example



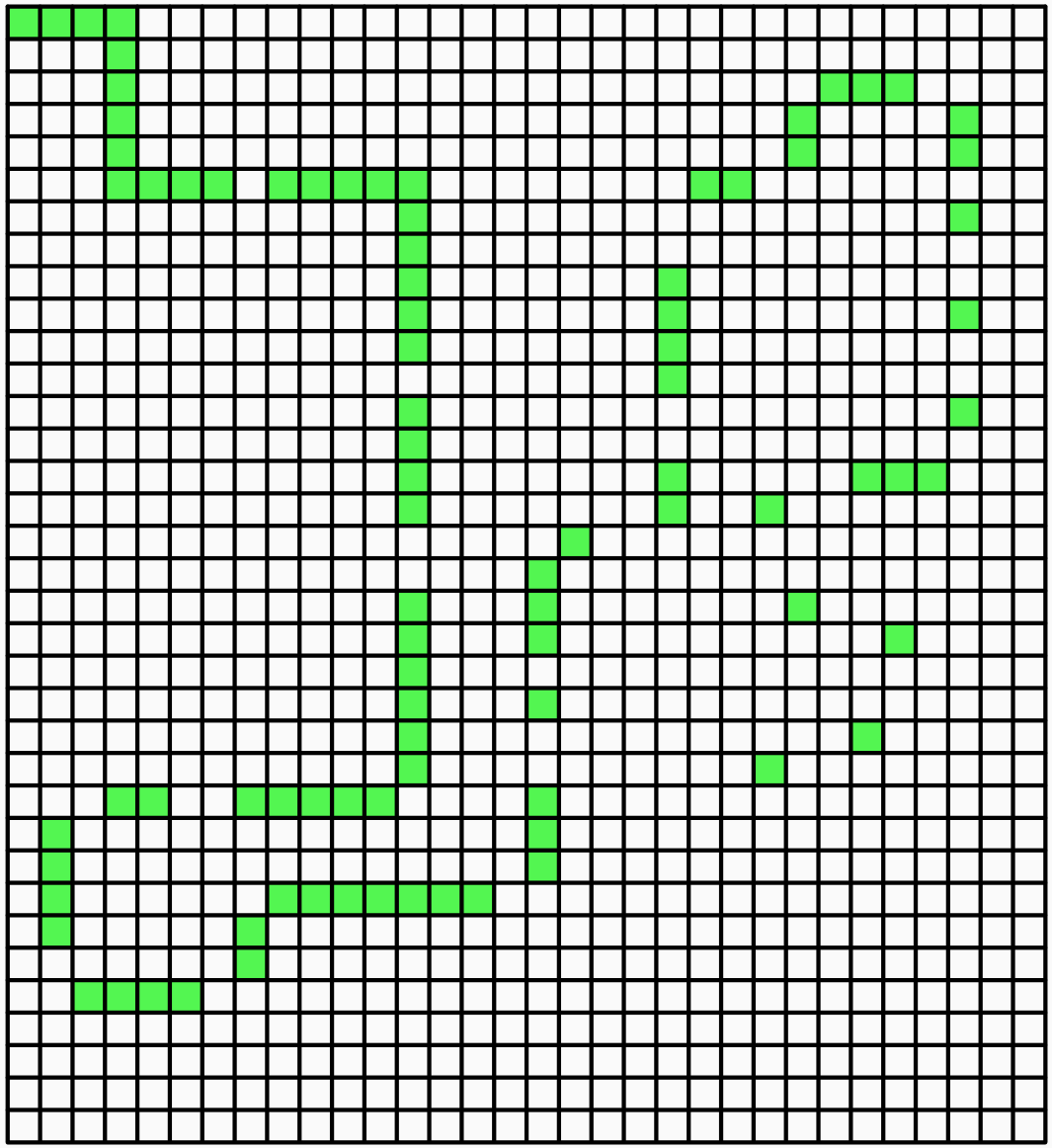


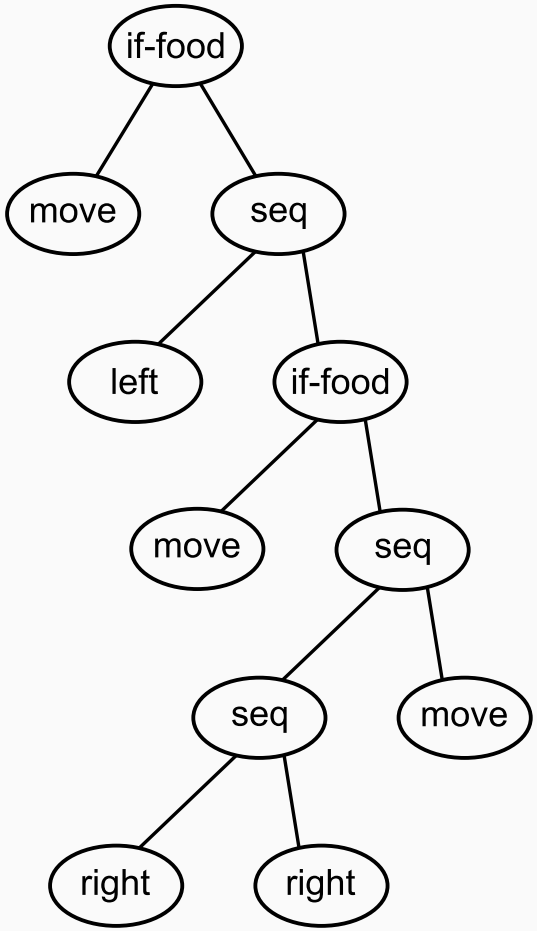
Example



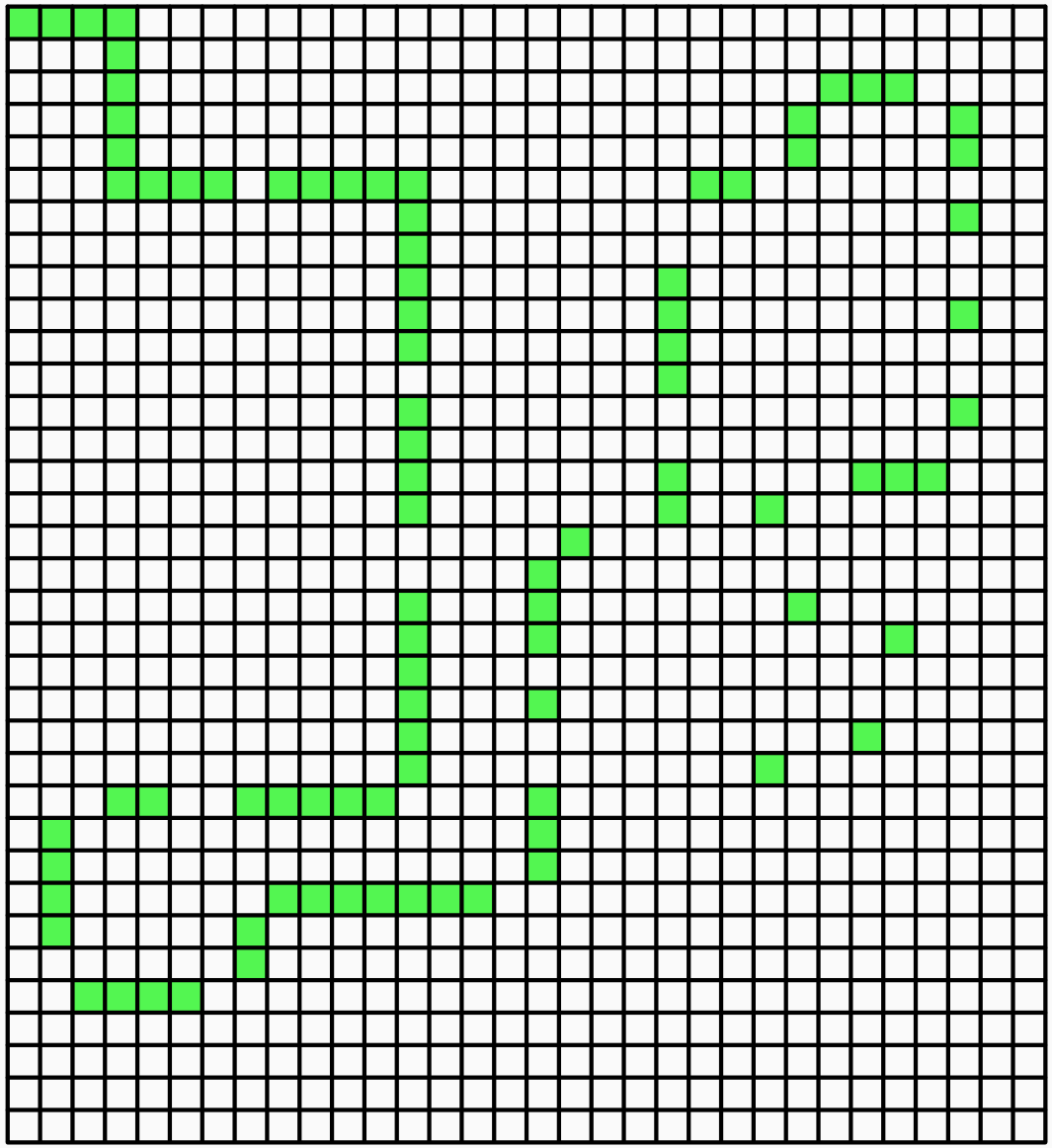


Example





Example



Genetic Programming

Summary

Summary

Genetic programming

- Uses evolution to design programs
- Individuals are programs or sequences of instructions
- Genotype is the code and phenotype is the program
- Comes with its own modifications and challenges

Tree Representation

- Encode program as a tree structure of operations
- Types:
  - Functional: for functions, nodes are operators or values
  - Imperative: for programs, nodes are instructions
- Most common representation

Application: Santa Fé trail

Learning Classifier Systems

Rule set

- We start from a set of possible rules
- A rule dictates an appropriate action
  - Based on the environment / input
  - Of the form `if (condition) then action`
- A rule set provides an action for each situation

if temp < target - 5 then heating max
if temp < target & target < 20 then heating med
- -

Learning Classifier Systems

Rule set

- We start from a set of possible rules
- A rule dictates an appropriate action
  - Based on the environment / input
  - Of the form `if (condition) then action`
- A rule set provides an action for each situation

Learning classifier systems

- Evolutionary algorithm that selects a set of rules
- Pittsburgh-style: individual is a rule set
- Michigan-style: individual is a rule, classifier is population

Encoding

Pittsburgh-style LCS

- Sequence of rule parameters
  - Each rule has condition, action and payoff
- Fitness is total payoff of actions taken for a data set

Michigan-style LCS

- Single rule parameters: condition, action and payoff
- Selection applied for a single chosen data point
  - Only rules that apply are evaluated for fitness
  - And only those get to recombine and mutate

Evolution Strategies

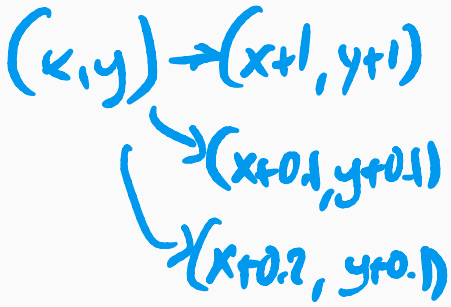
Evolution Strategies

Evolutionary algorithms for real values

- Similar to genetic algorithms
- Use real values for the components of the individual
- Typically $(\mu, 4\mu)$-selection
  - Generate $4n$ children, take n best

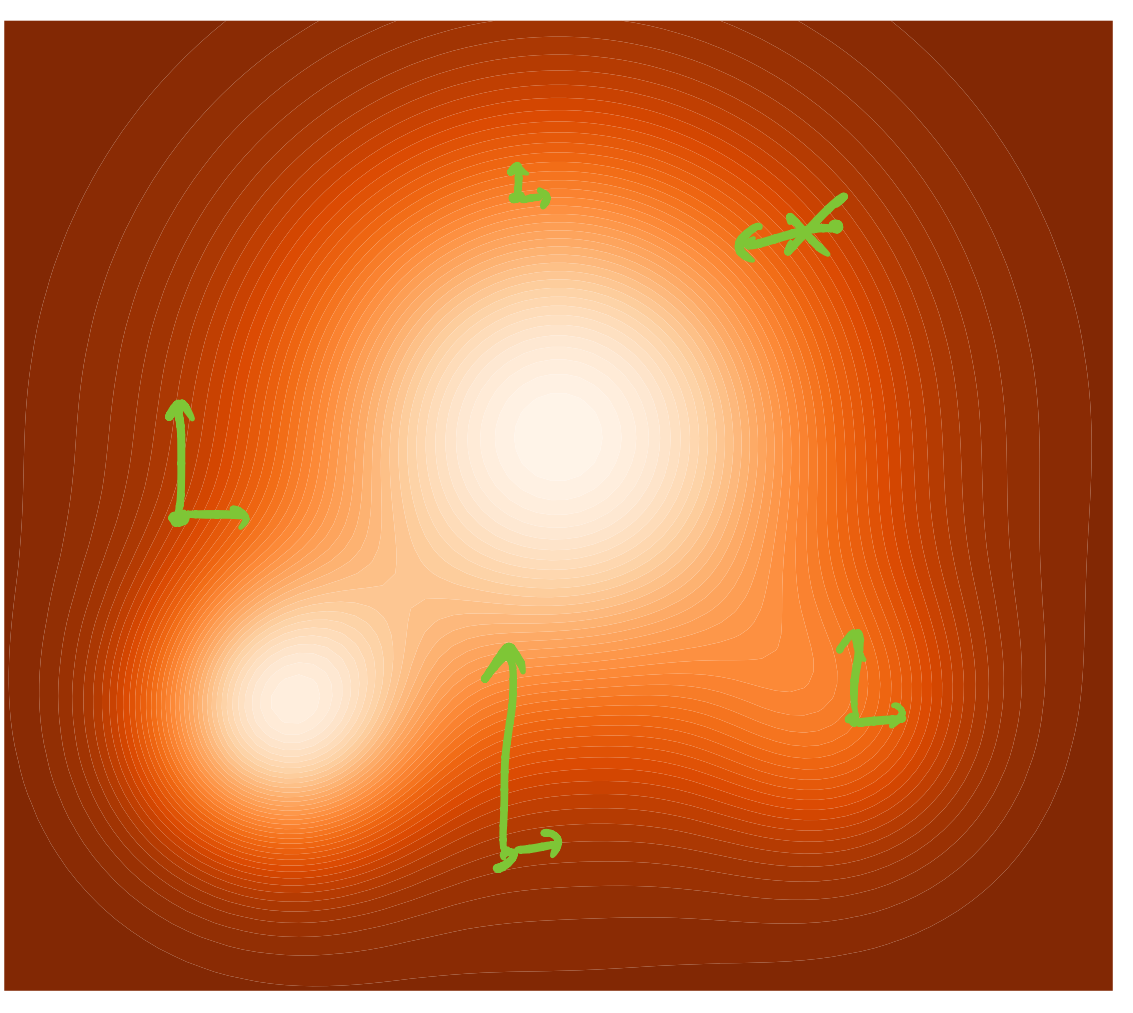
Self-adaptation of strategy parameters

- Parameters are included in the genome
  - And evolve with the individuals
- Includes mutation distance for each component



Example

Minimize a function



Encoding

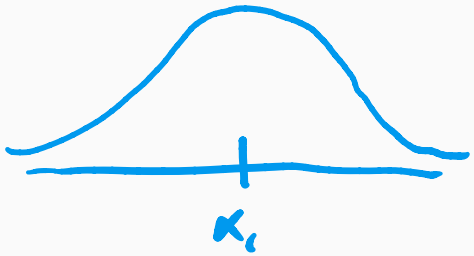
Values

- Individuals are vectors of reals
- Usually encoded as floating-point numbers

Parameters

- Self-adapting parameters are also in the genome
- Stored similarly to the values

Variation Operators



Mutation

- Add Gaussian noise with average 0, standard deviation σ_i
  - σ_i given as a parameter

Recombination: combine individuals component-wise

- Discrete: take either at random
  - Typically used for values
- Intermediate: average values of both parents
  - Used for parameters

Conclusion

Summary

Genetic programming

- Individuals are programs or sequences of instructions
- Usually represented as trees of operations
- Applications to robot behavior, design creatures, ...

Learning Classifier Systems

- Start from a defined set of rules
- Evolve a good rule set for any situation

Evolution strategies

- Evolutionary algorithms for vectors of reals
- Self-adaptation of mutation parameters