CS 581

Advanced Artificial Intelligence

February 12, 2024

Announcements / Reminders

- Please follow the Week 05 To Do List instructions (if you haven't already)
- Written Assignment #02: due on Monday 02/19 at 11:59 PM CST
- Programming Assignment #01: due on Sunday 03/03 at 11:59 PM CST

- **Midterm Exam: 02/21/2024**
  - Section 02 – Make arrangements with Mr. Charles Scott
  - <u>WE WILL HAVE OUR EXAM IN A DIFFERENT ROOM</u>

Midterm Exam: Rules

- Exam will be pen and paper
- No electronic devices allowed
  - including AirPods, earbuds, etc.
  - exception: REGULAR calculator (not a phone app) if you need it
  - all your electronic devices need to be hidden from view
- No communication allowed
- Closed book / closed notes
  - you can bring ONE letter-sized double-sided cheat sheet
- NO programming will be involved, however you are expected to understand algorithms to work out solutions by hand
- Material: everything I covered in class without saying “this is not going to be on the exam” is fair game

Midterm Exam: Essentials

Chapter 2 Intelligent Agents (and corresponding lecture slides):

- **Section 2.1: understand the terminology and concepts.**
- **Section 2.2: understand the terminology and concepts.**
- **Section 2.3: understand the terminology and concepts.**
  - **Be comfortable with the PEAS description and environment properties.**
- **Section 2.4: understand the differences between different agent types and how they affect your design choices.**
  - **You may be asked to pick the best agent type for some problem and justify your answer.**
- **Go through the chapter summary.**

Midterm Exam: Essentials

Chapter 3 Solving Problems by Search (and corresponding lecture slides):

- **Section 3.1: understand the terminology and concepts.**
  - **Be comfortable with defining a search problem.**
- **Section 3.2: go through examples**
- **Section 3.3 and 3.4: understand the terminology and concepts.**
  - **Ignore sections 3.4.4 and 3.4.5 for the exam.**
- **Section 3.5: understand the terminology and concepts / algorithms. You may be asked to solve a search problem by hand.**

- **Section 3.6: review the introduction to this section.**
- **Go through the chapter summary. FOCUS ON A* algorithm**

Midterm Exam: Essentials

Chapter 4 Search in Complex Environments (and corresponding lecture slides):

- **Local Search and Optimization Problems**
  - **4.1.1 Hill-climbing search**
  - **4.1.2 Simulated annealing**
  - **4.1.4 Evolutionary algorithms**
- **IGNORE TABU SEARCH**
- **…and everything related to Evolutionary algorithms that I covered in class (especially: EVERYTHING about GENETIC ALGORITHM)**

Midterm Exam: Essentials

Chapter 5 Adversarial Search and Games (and corresponding lecture slides):

- **Section 5.1: understand the terminology and concepts.**
- **Section 5.2: understand the terminology and concepts.**
  - **You may be asked to solve an adversarial problem by hand using Min-Max and alpha-beta pruning. Ignore section 5.2.2.**
- **Go through the chapter summary.**

Midterm Exam: Essentials

Chapter 6 Constraint Satisfaction Problems (and corresponding lecture slides):

- **Section 6.1: understand the terminology and concepts.**
  - **You may be asked to formally define a constraint satisfaction problem.**
- **Section 6.2: understand the terminology and concepts.**
  - **You may be asked to apply techniques from this chapter. Ignore sections 6.2.4 and 6.2.5.**
- **Section 6.3: understand the terminology and concepts.**
  - **You may be asked to apply techniques from this chapter. Ignore sections 6.3.3 and 6.3.4.**
- **Go through the chapter summary.**

Midterm Exam: Essentials

Otherwise:

- **Everything that I will cover this week during 02/12 and 02/14 sessions (unless I say “not on the exam”)**

Plan for Today

- Solving problems by Searching
  - Evolutionary Algorithms
    - Genetic Algorithm – more on fitness
    - Genetic Programming
    - Other
  - Genetic Fuzzy Systems
  - Ant Colony Optimization Algorithm

Genetic Algorithm: Fitness Function

- Has to be clearly defined and implemented efficiently.
  - cannot be a bottleneck .
- Should quantitatively measure how fit a given solution is in solving the problem.
- Should produce intuitive results.
  - The best/worst should be scored accordingly
- Use Σ and $\prod$ of components (A, 1/B, etc.)
  - normalize / rescale components
  - use distance measures
  - introduce pentalties for violating ("straying away") constraints

Evolutionary Algorithms: Speciation

- Nature: speciation occurs when two similar reproducing beings evolve to become too dissimilar to share genetic information effectively or correctly.
  - they are incapable of mating to produce offspring.
    - Example: a horse and a donkey mating to produce a mule. However in this case the Mule is usually infertile, and so the genetic isolation of the two parent species is maintained.
- Implementation: speciation → some mathematical function that establishes the similarity between two candidate solutions in the population
  - If the result of the similarity is too low, the crossover operator is disallowed between those individuals.

Genetic Programming

Traditional Programming vs …

Traditional programming:

Input data + Program (Rules) = Output

…:

Input data + Output = Program (Rules)

Genetic Programming

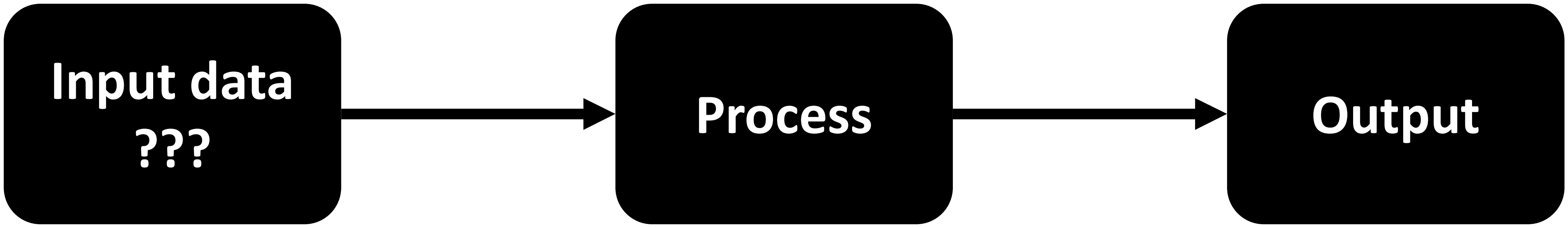
- **Genetic programming (GP) is an automated method for creating a working computer program from a high-level problem statement of a problem.**
- **Genetic programming starts from a high-level statement of “what needs to be done” and automatically creates a computer program to solve the problem**
- **John Koza - "Genetic Programming: On the Programming of Computers by Means of Natural Selection"**

Genetic Programming

- Goal: find a program that generates correct solution
  - based on input – correct output data
- Genotype: tree [GA: string]
- Phenotype: actual program [GA: f() value)]
- Evaluation:
  - run program
  - see if output data is expected

GA vs GP

Genetic Algorithm:



Genetic Programming:

Genetic Programming: Preparation

- Determine the set of terminals.
- Select the set of primitive functions.
- Define the fitness function.
- Decide on the parameters for controlling the run.
- Choose the method for designating a result of the run.

Problem Description

Objective	Find a computer program with one input X for which the output Y is equal to the given data
Terminal set	$T = \{\text{variables, constants}\}$
Function/operator set	$F = \{\text{functions, operators}\}$
Initial population	Randomly created individuals built using elements from T and F.
Fitness function	$\lvert y_0' - y_0\rvert + \lvert y_1' - y_1\rvert + \ldots$ where y_i' is computed output and y_i is given output for x_i in the range $[-1,1]$
Termination condition	An individual emerges with the value of its fitness function is less than ε

Subtree Crossover

Consider a program in C programming language:

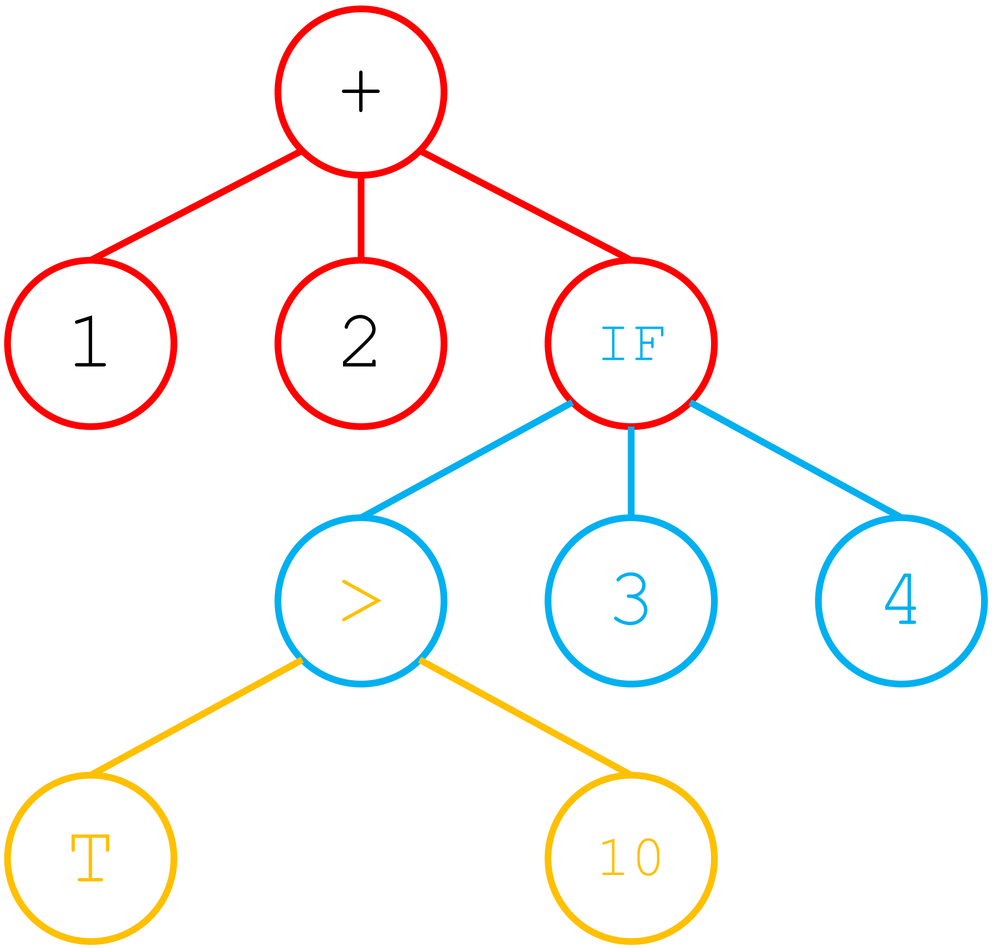
```
int foo (int time)
{
   int temp1, temp2;
   if (T > 10)
       temp1 = 3;
   else
       temp1 = 4;
   temp2 = temp1 + 1 + 2;
   return (temp2);
}
```

Equivalent expression:

```
(+ 1 2 (IF (> T 10) 3 4))
```

Program Tree

(+ 1 2 (IF (> T 10) 3 4))

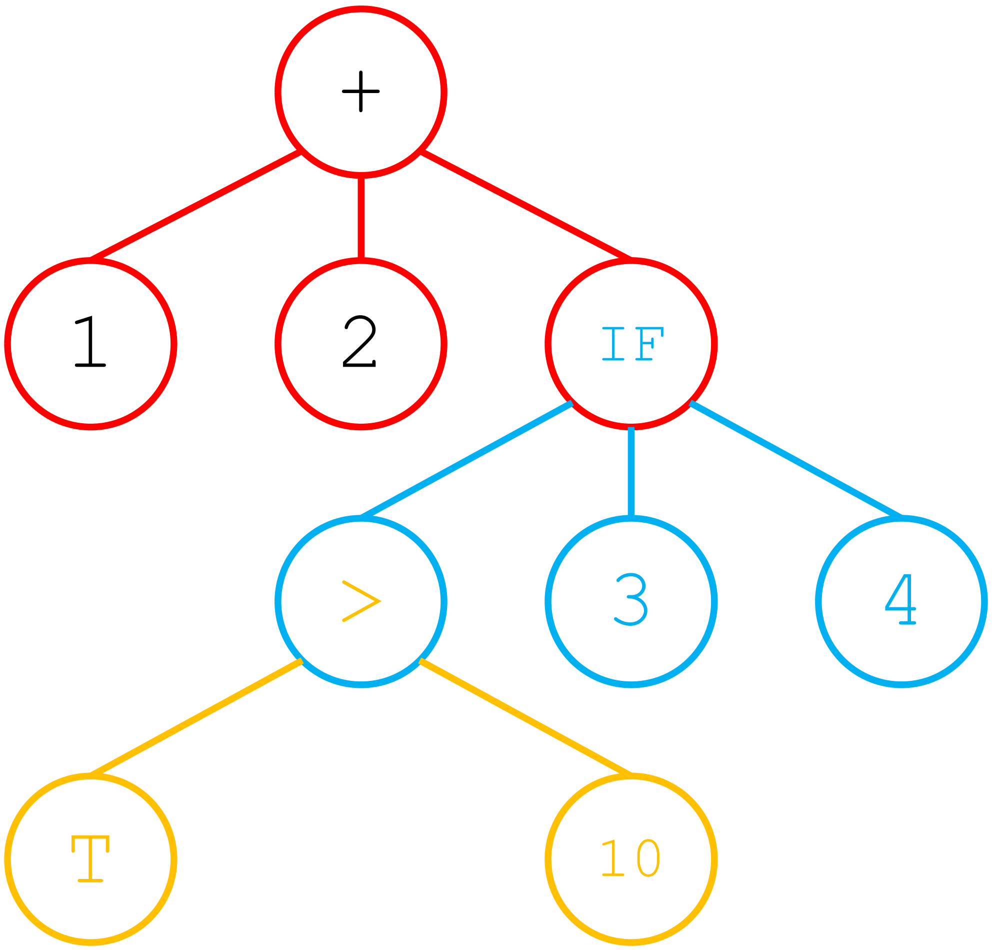


Functions/Operators vs. Terminals

- **Functions / Operators**
  - **require arguments**
- **Terminals:**
  - **tree leaf**
  - **constants**
  - **variables**
  - **“external” function calls**

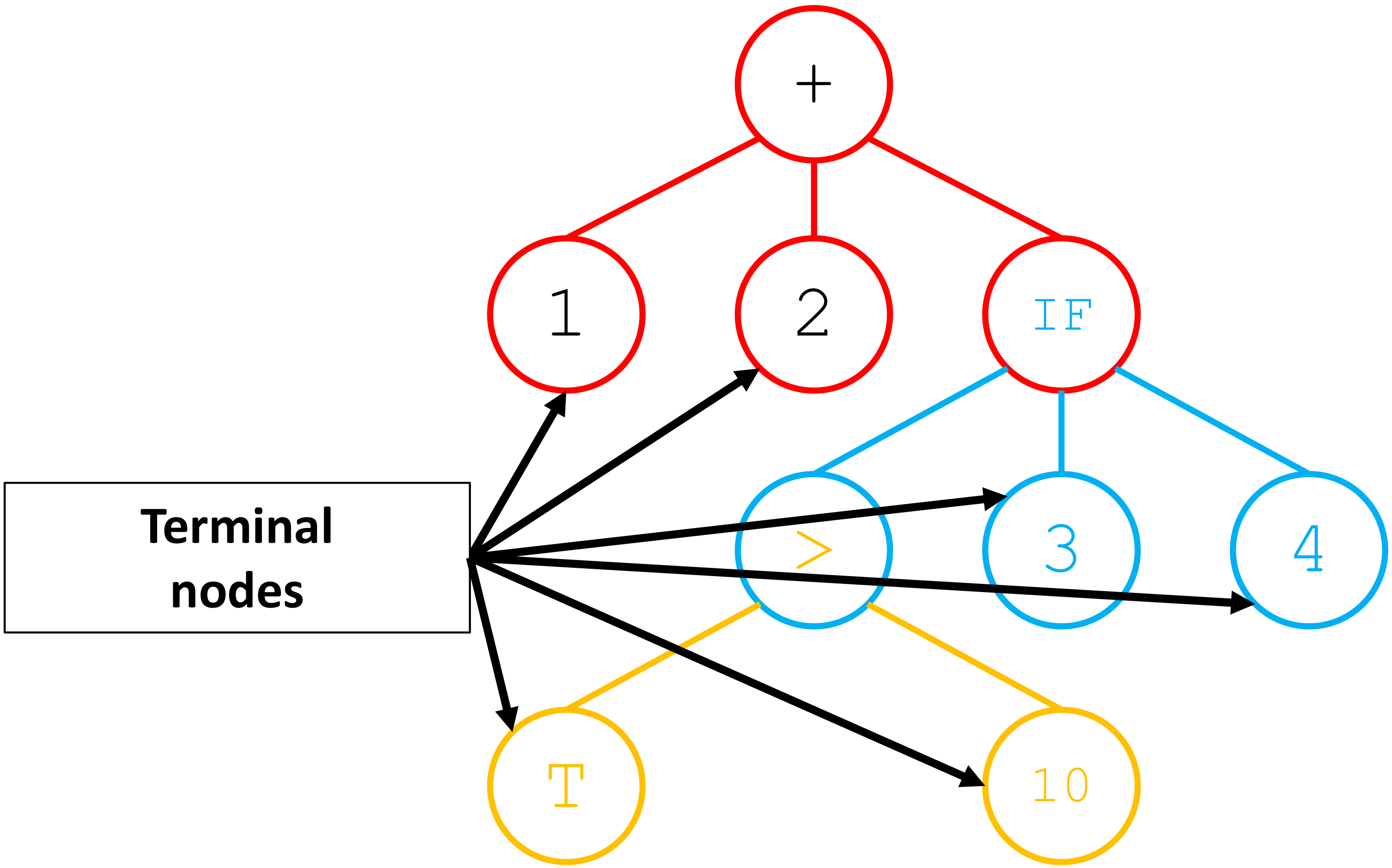
Program Tree

`(+ 1 2 (IF (> T 10) 3 4))`



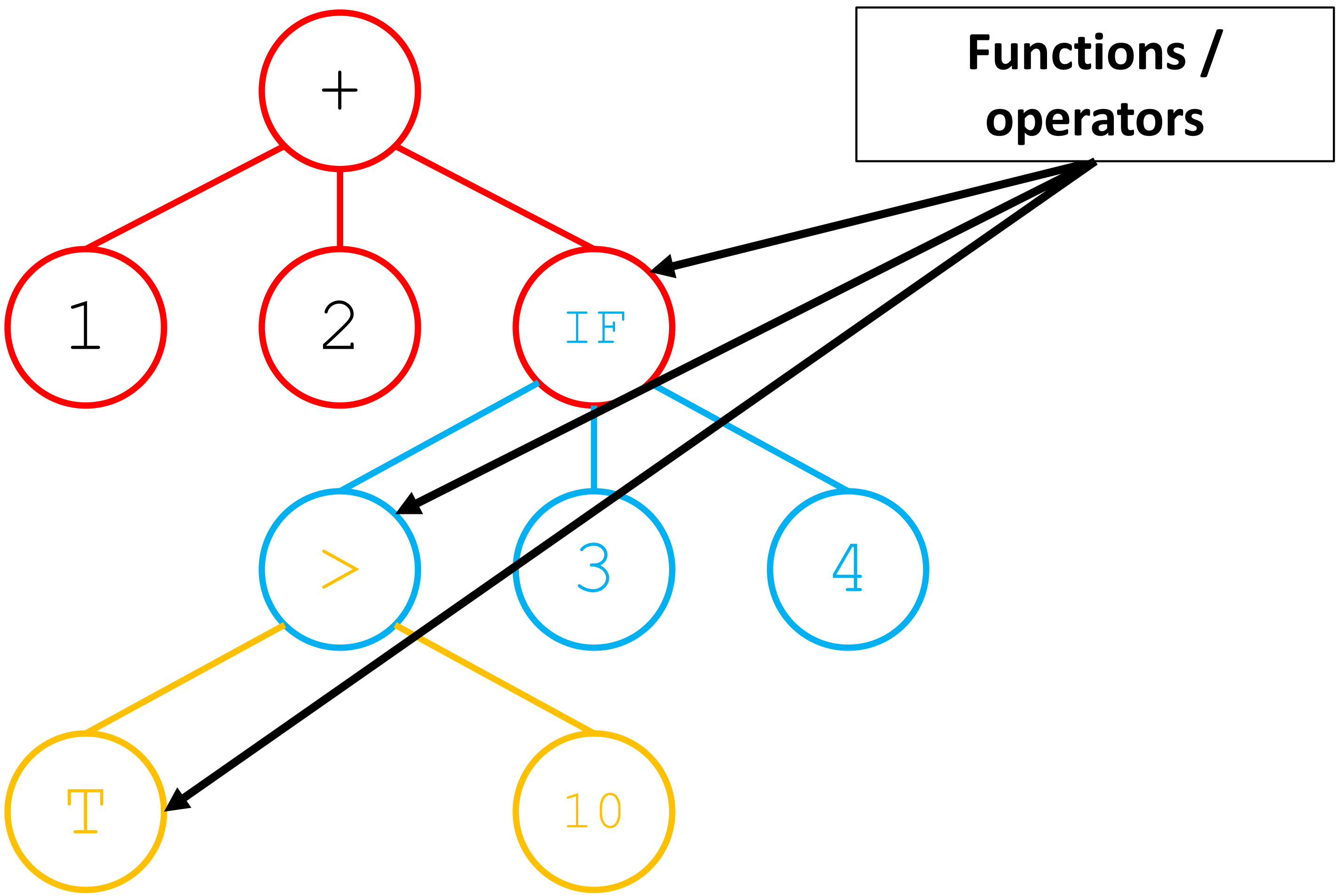
Program Tree

`(+ 1 2 (IF (> T 10) 3 4))`



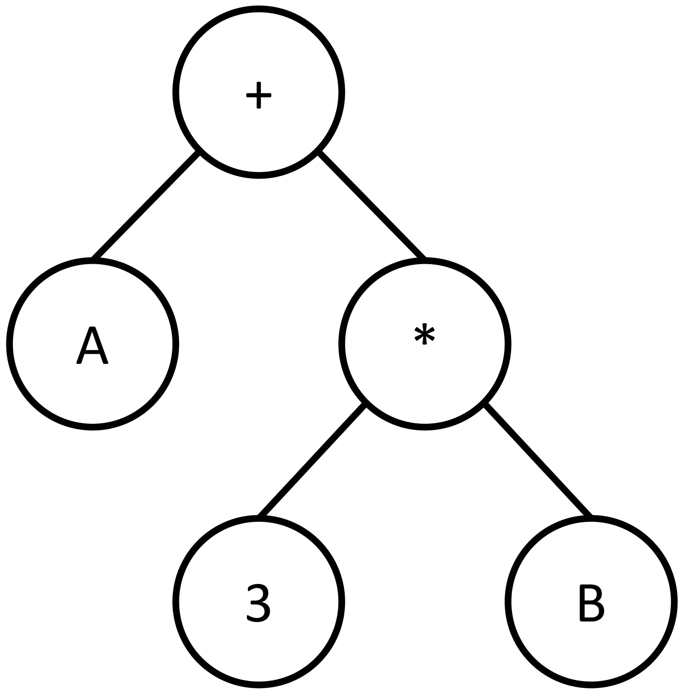
Program Tree

`(+ 1 2 (IF (> T 10) 3 4))`



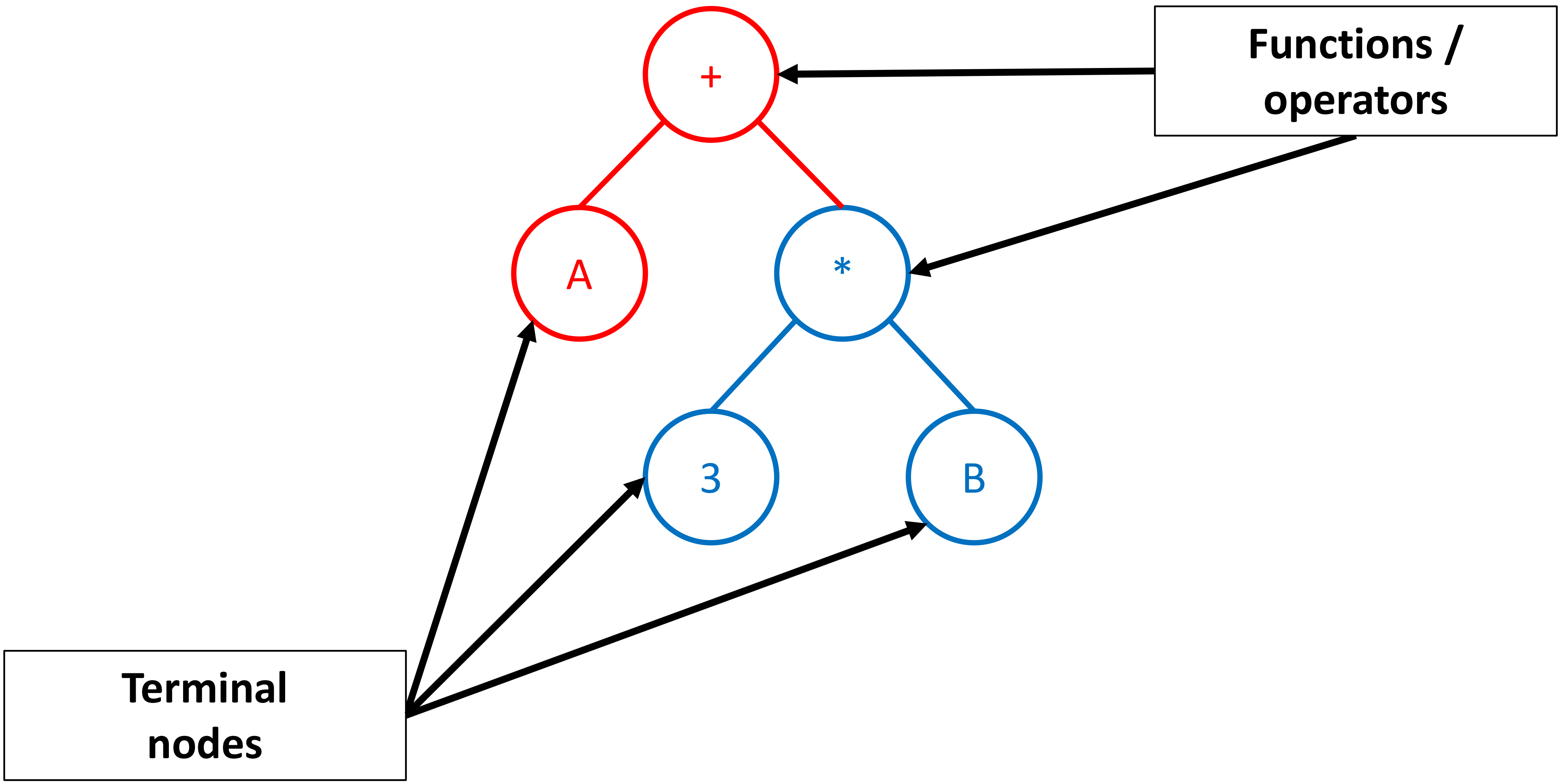
Abstract Syntax Trees

A + 3 * B



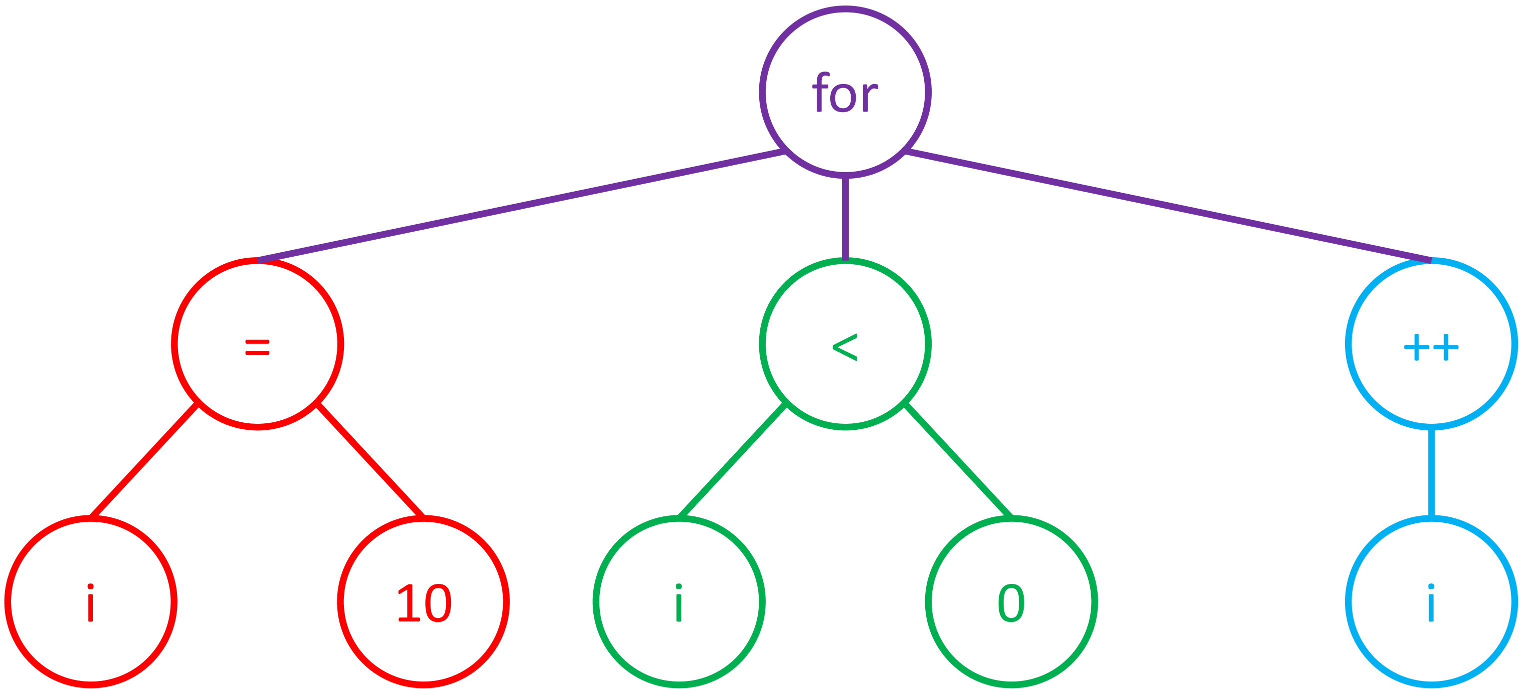
Abstract Syntax Trees

A + 3 * B = A + (3 * B) = (A + (3 * B))



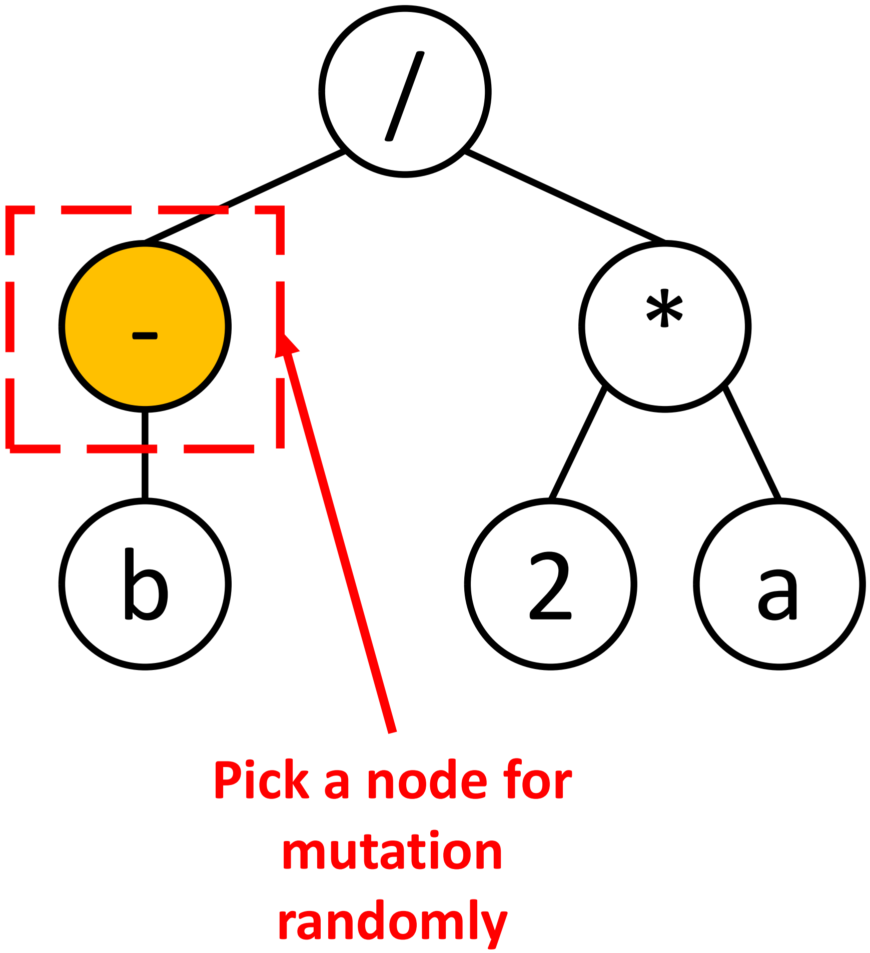
Abstract Syntax Trees

for(i = 0; i < 0; i++)



Genetic Programming: Mutation

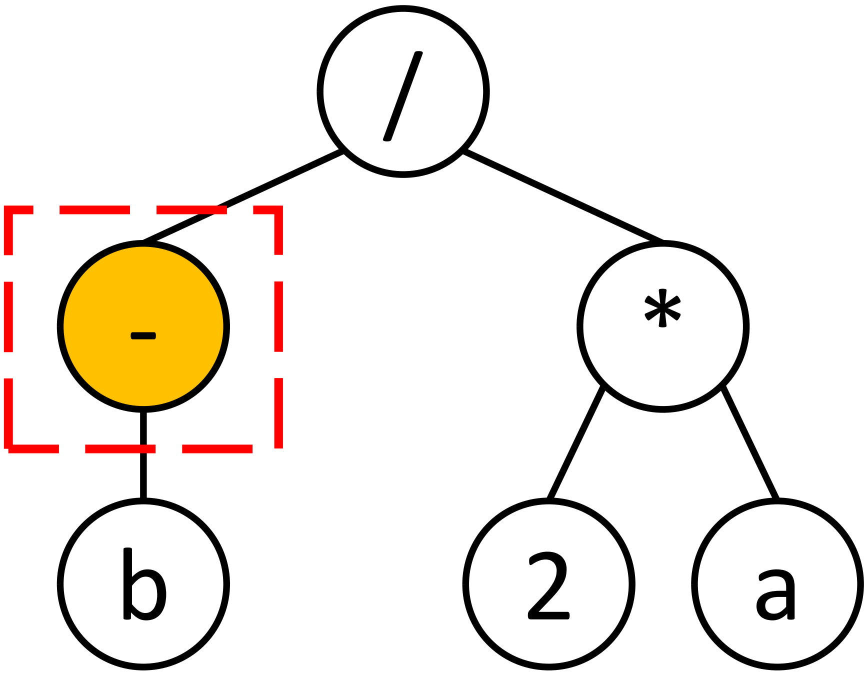
$$\frac{-b}{2 * a}$$



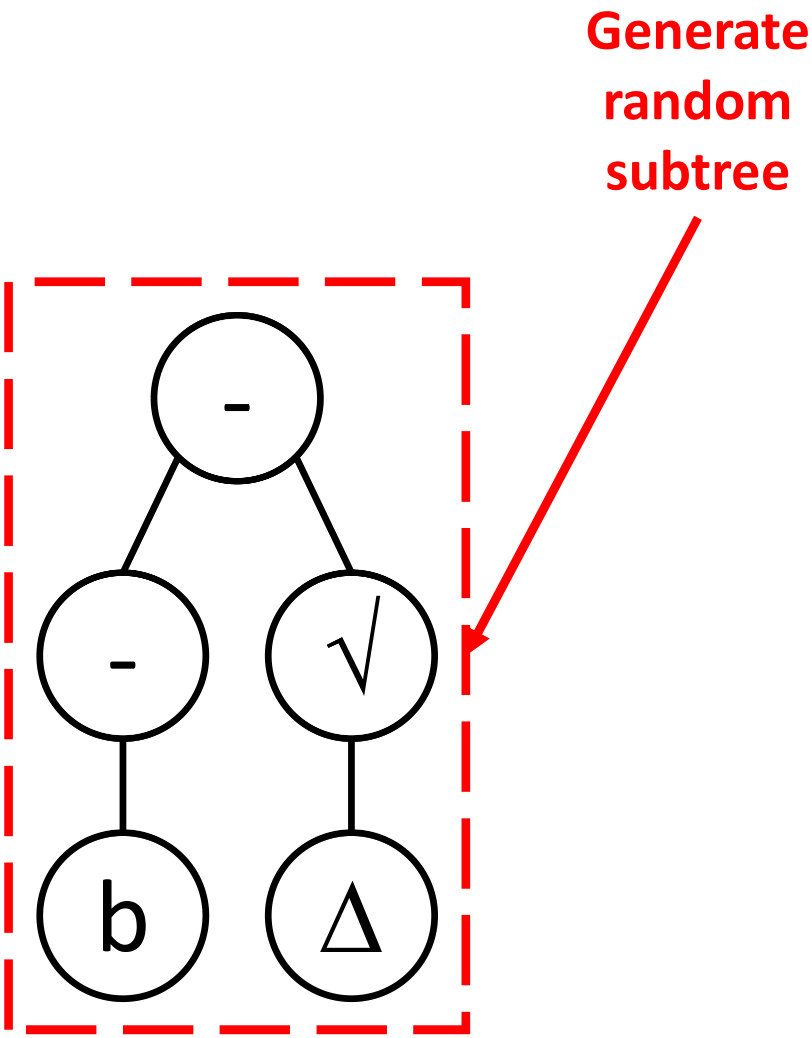
Individual B

Genetic Programming: Mutation

$$\frac{-b}{2 * a}$$

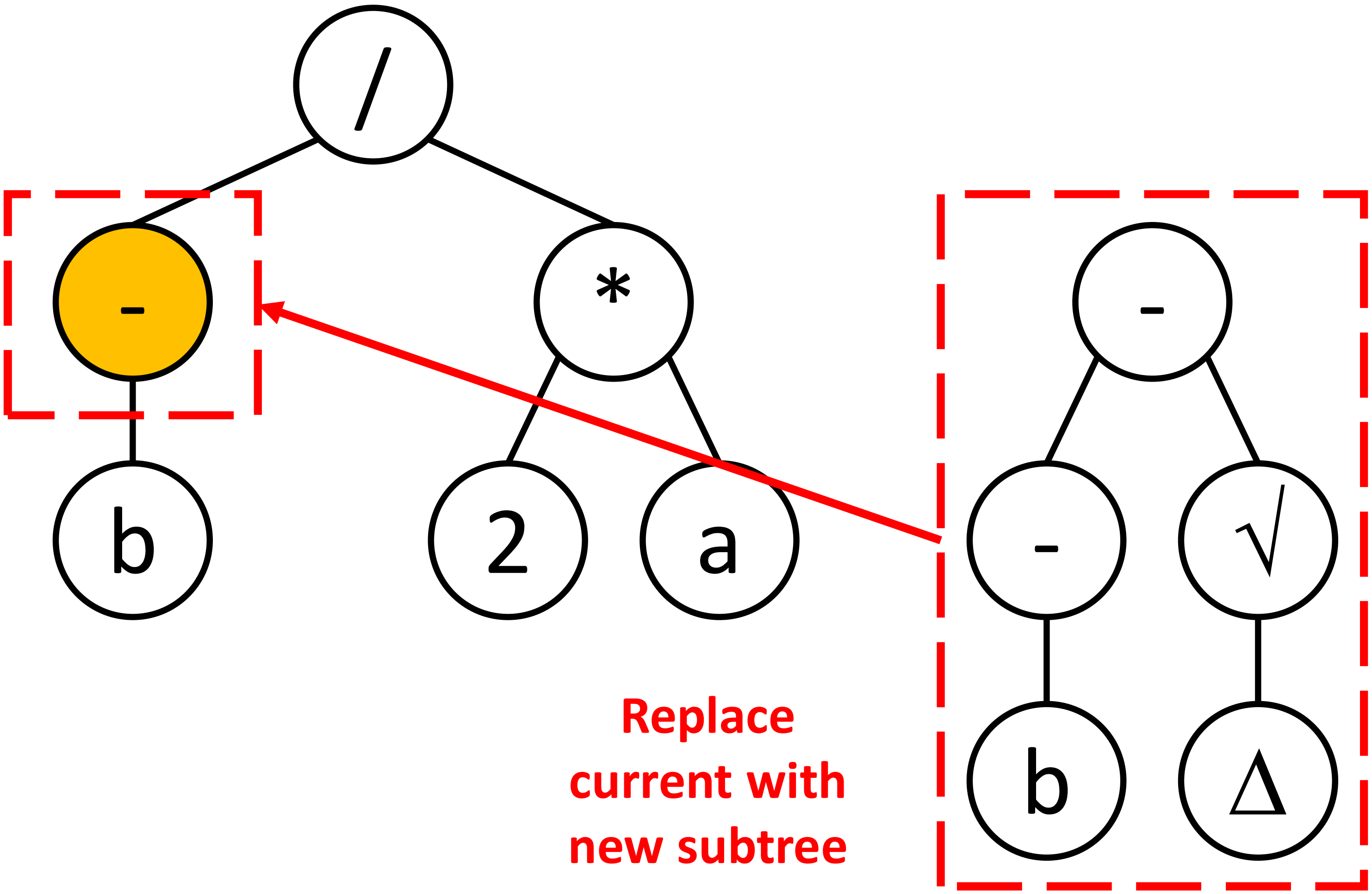


Individual B



Genetic Programming: Mutation

$$\frac{-b}{2 * a}$$

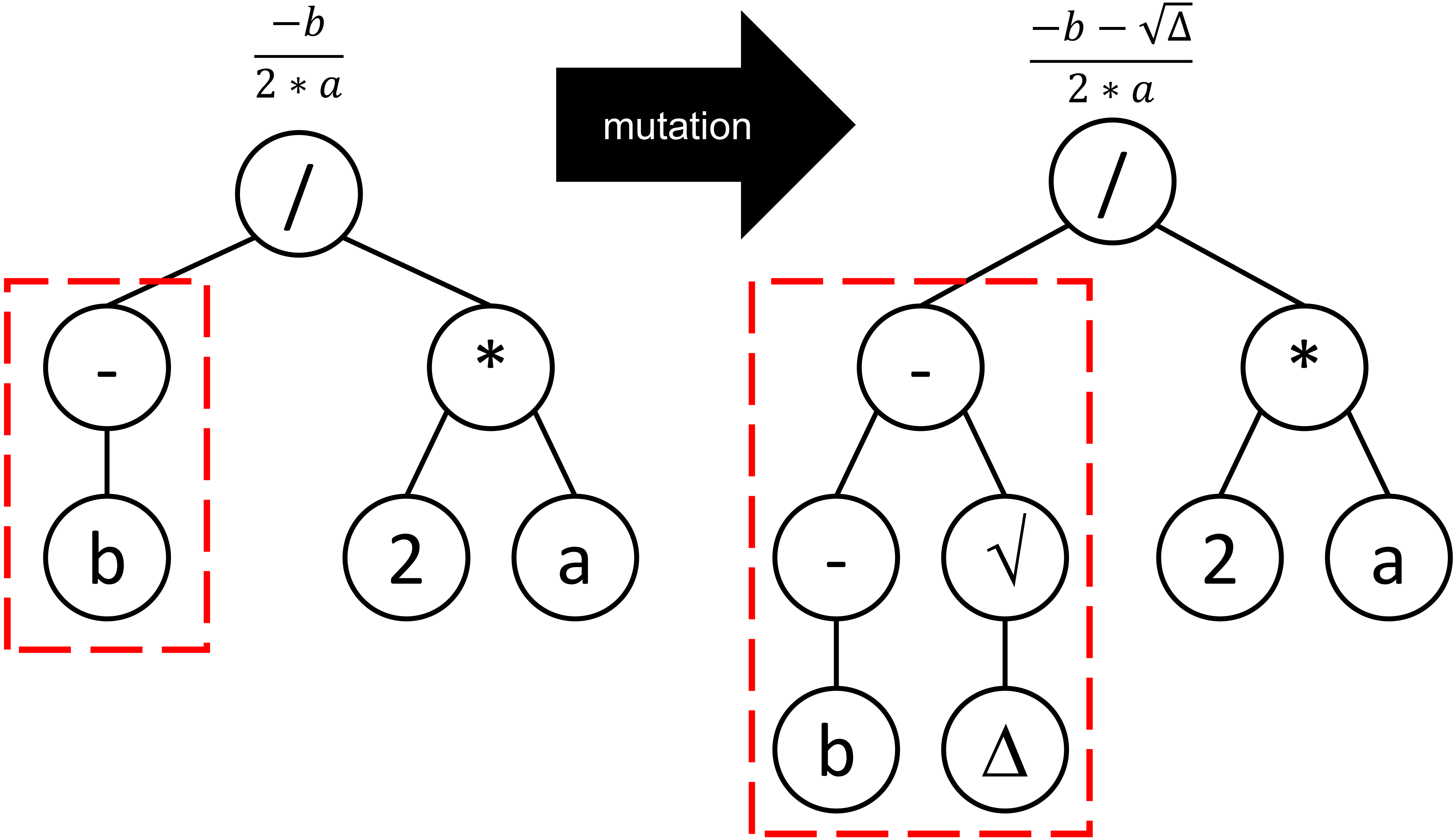


Individual B

Genetic Programming: Mutation

$$\frac{-b}{2*a}$$

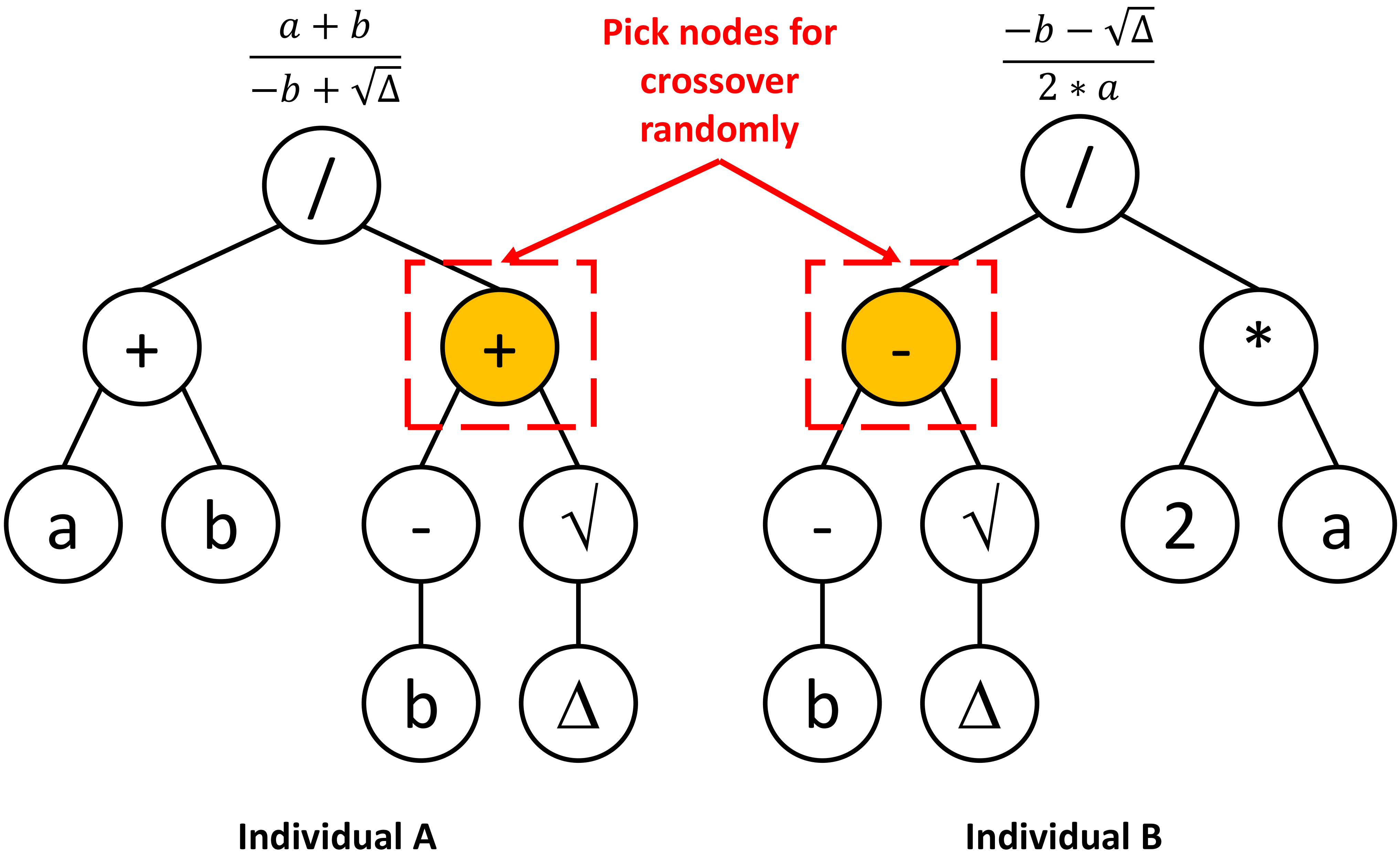
$$\frac{-b-\sqrt{\Delta}}{2*a}$$



Individual B

Individual B after mutation

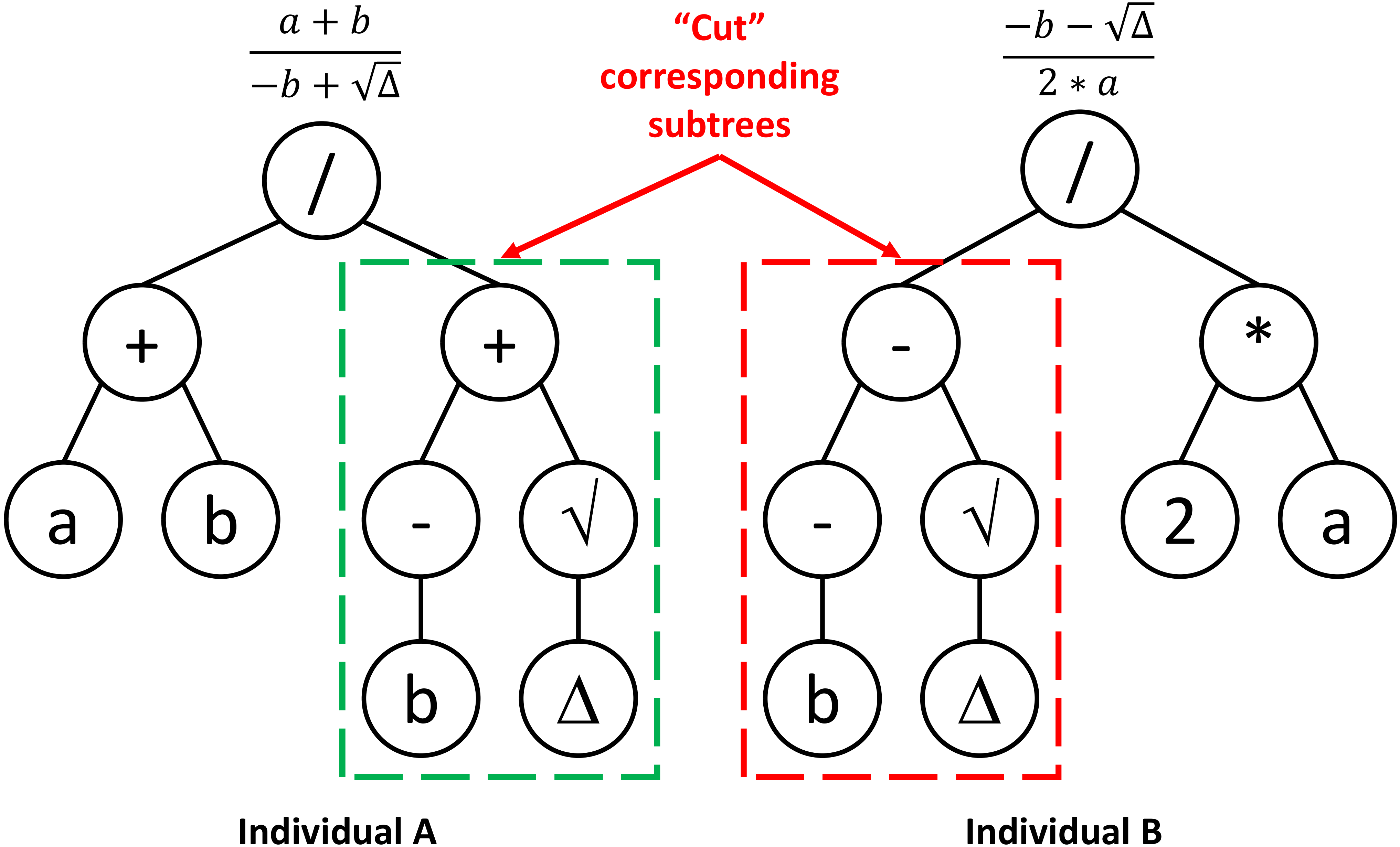
Genetic Programming: Crossover



Genetic Programming: Crossover

$$\frac{a+b}{-b+\sqrt{\Delta}}$$

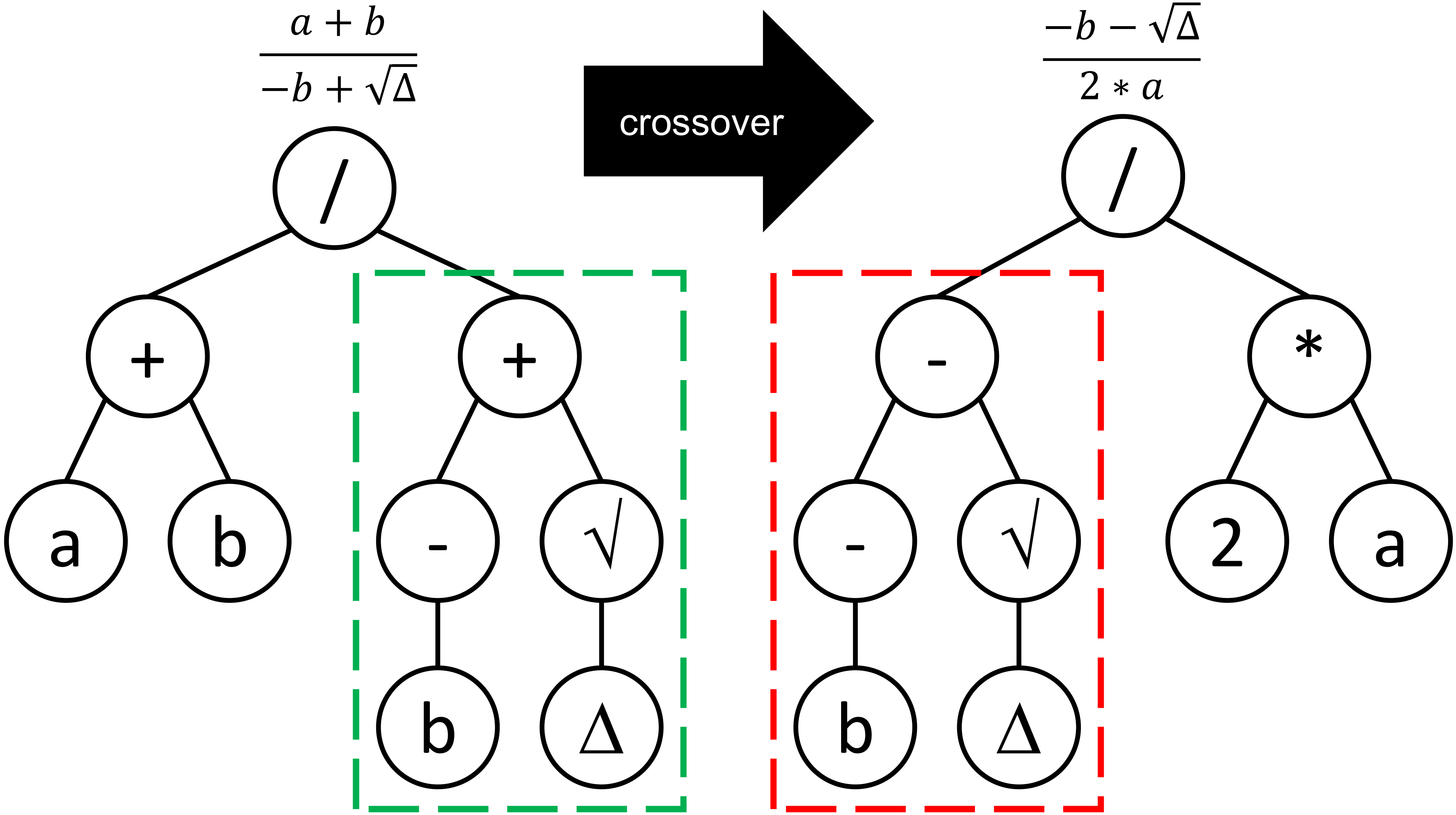
$$\frac{-b-\sqrt{\Delta}}{2*a}$$



Genetic Programming: Crossover

$$\frac{a+b}{-b+\sqrt{\Delta}}$$

$$\frac{-b-\sqrt{\Delta}}{2*a}$$

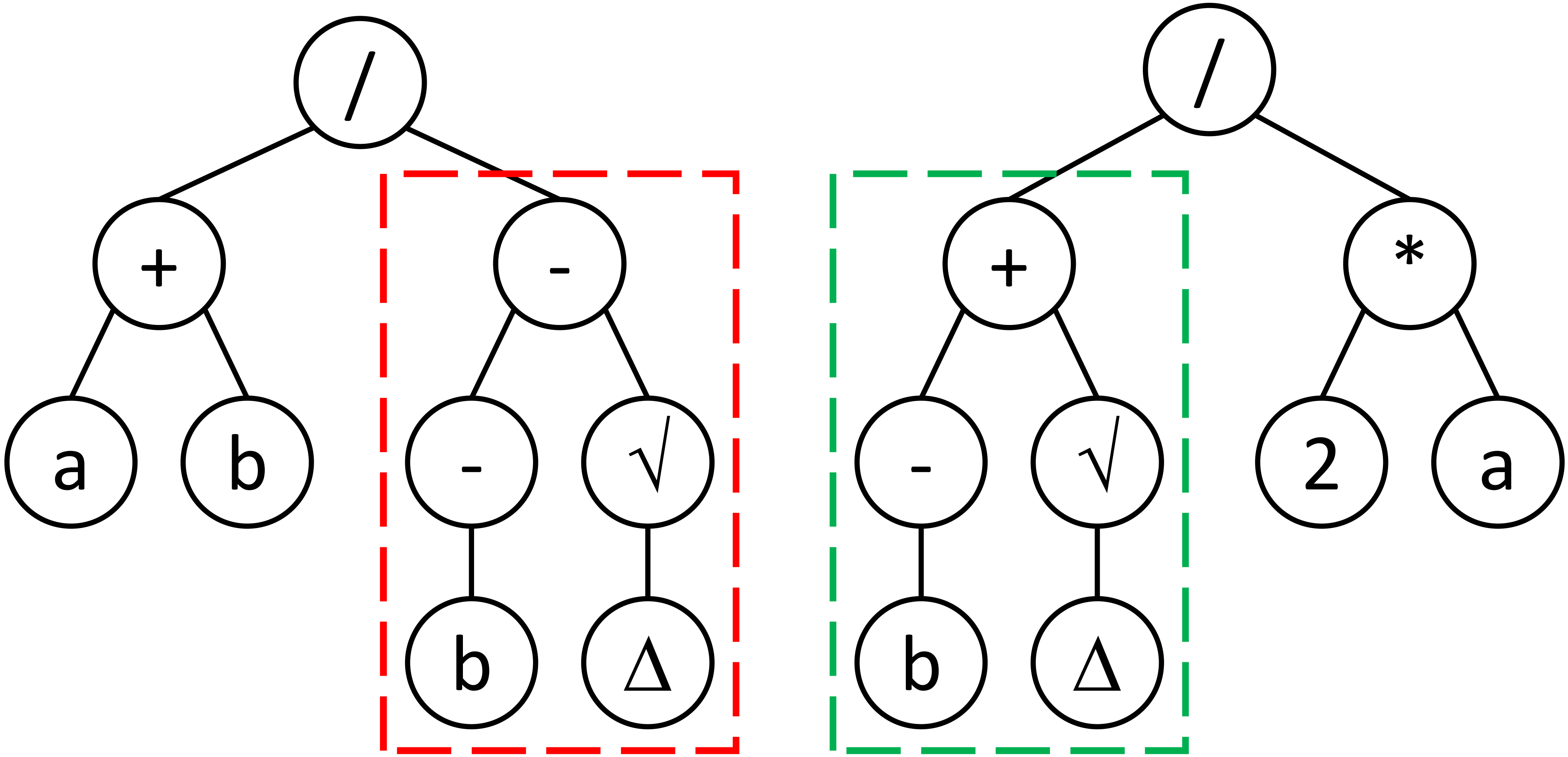


Individual A **Individual B**

Genetic Programming: Crossover

$$\frac{a+b}{-b-\sqrt{\Delta}}$$

$$\frac{-b+\sqrt{\Delta}}{2*a}$$



Child 1 of individuals A and B

Child 2 of individuals A and B

Example Problem: Data

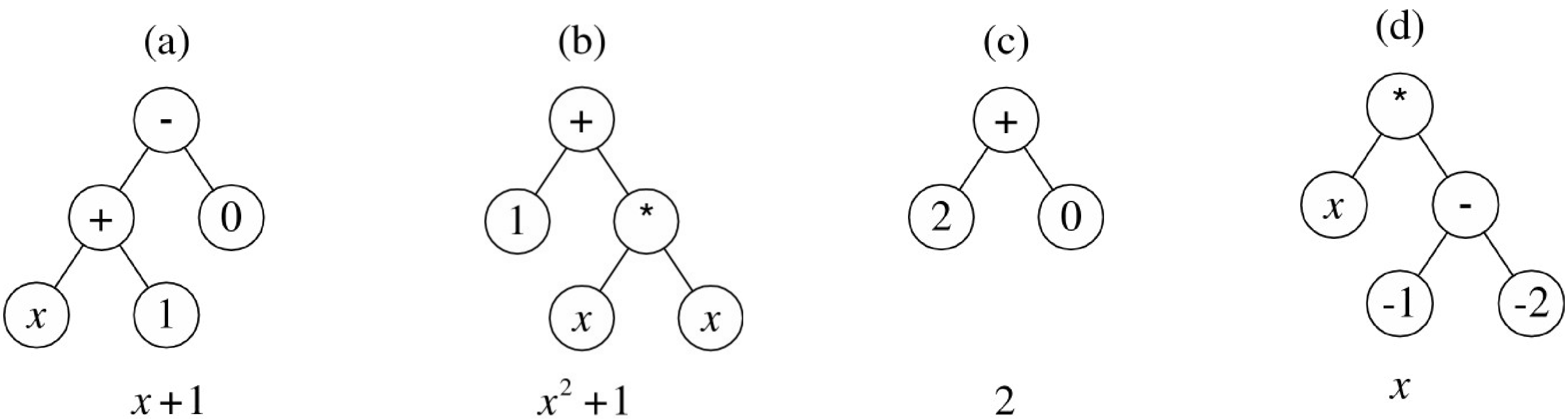
Input: Independent variable X	Output: Dependent variable Y
-1.00	1.00
-0.80	0.84
-0.60	0.76
-0.40	0.76
-0.20	0.84
0.00	1.00
0.20	1.24
0.40	1.56
0.60	1.96
0.80	2.44
1.00	3.00

Example Problem Description

Objective	Find a computer program with one input X for which the output Y is equal to the given data
Terminal set	$T = \{X, \text{Constants}\}$
Function/operator set	$F = \{+, -, *, /\}$
Initial population	Randomly created individuals built using elements from T and F.
Fitness function	$\lvert y_0' - y_0 \rvert + \lvert y_1' - y_1 \rvert + \ldots$ where y_i' is computed output and y_i is given output for x_i in the range [-1,1]
Termination condition	An individual emerges with the value of its fitness function is less than ε

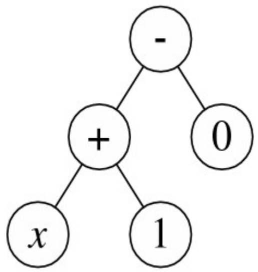
Example Problem: Generation 0

Initial population of four randomly created individuals:



Example Problem: Generation 1

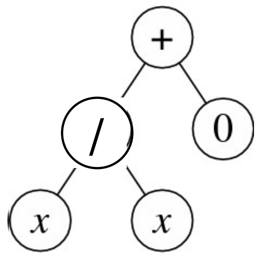
(a)



$x+1$

Copy of (a)

(b)

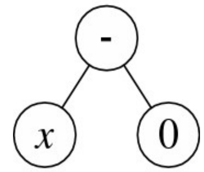


1

Mutant of (c)

picking “2” as mutation point

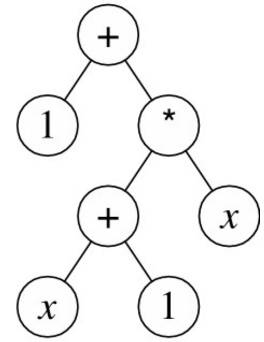
(c)



x

First offspring of crossover of (a) and (b) picking “+” of parent (a) and left-most “x” of parent (b) as crossover points

(d)



x^2+x+1

Second offspring of crossover of (a) and (b) picking “+” of parent (a) and left-most “x” of parent (b) as crossover points

Genetic Programming: Application

Science | Current Issue | First release papers | Archive | About | Submit manus

HOME > SCIENCE > VOL. 324, NO. 5923 > DISTILLING FREE-FORM NATURAL LAWS FROM EXPERIMENTAL DATA

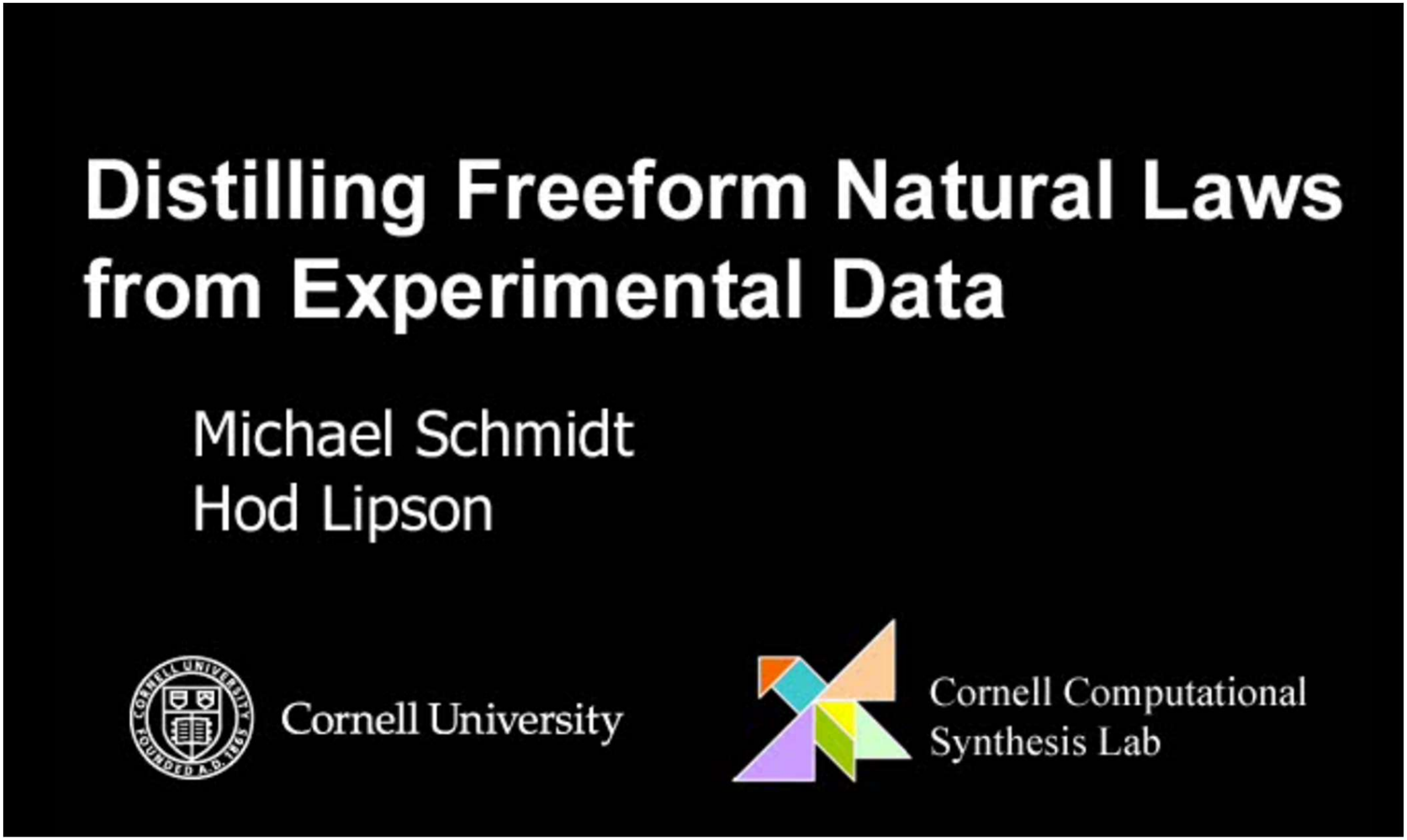
REPORTS

Distilling Free-Form Natural Laws from Experimental Data

MICHAEL SCHMIDT AND HOD LIPSON Authors Info & Affiliations

SCIENCE • 3 Apr 2009 • Vol 324, Issue 5923 • pp. 81-85 • DOI: 10.1126/science.1165893

Genetic Programming: Video



Other (Key) Evolutionary Approaches

Evolutionary Programming

- Similar to genetic programming, but the solution is a set parameters for a predefined fixed computer program, not a generated computer program

- Solution fitness is determined by how well the fixed computer program performs based on the parameters encoded in an individual

Evolutionary Strategies

Species DNA structure

Gene 1			Gene 2			Gene 3			

Genetic Algorithm

Gene 1			Gene 2			Gene 3			
0	1	0	1	1	0	1	0	1	1

Solution

Strategy

Evolutionary Strategies

Gene 1			Gene 2			Gene 3			
1	0	1	1	1	0	1	0	1	1

Evolutionary Algorithms ARE USED!

arXiv > cs > arXiv:1712.06567

Search... All fields Search

Help | Advanced Search

Computer Science > Neural and Evolutionary Computing

[Submitted on 18 Dec 2017 (v1), last revised 20 Apr 2018 (this version, v3)]

Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, Jeff Clune

Deep artificial neural networks (DNNs) are typically trained via gradient-based learning algorithms, namely backpropagation. Evolution strategies (ES) can rival backprop-based algorithms such as Q-learning and policy gradients on challenging deep reinforcement learning (RL) problems. However, ES can be considered a gradient-based algorithm because it performs stochastic gradient descent via an operation similar to a finite-difference approximation of the gradient. That raises the question of whether non-gradient-based evolutionary algorithms can work at DNN scales. Here we demonstrate they can: we evolve the weights of a DNN with a simple, gradient-free, population-based genetic algorithm (GA) and it performs well on hard deep RL problems, including Atari and humanoid locomotion. The Deep GA successfully evolves networks with over four million free parameters, the largest neural networks ever evolved with a traditional evolutionary algorithm. These results (1) expand our sense of the scale at which GAs can operate, (2) suggest intriguingly that in some cases following the gradient is not the best choice for optimizing performance, and (3) make immediately available the multitude of neuroevolution techniques that improve performance. We demonstrate the latter by showing that combining DNNs with novelty search, which encourages exploration on tasks with deceptive or sparse reward functions, can solve a high-dimensional problem on which reward-maximizing algorithms (e.g.\ DQN, A3C, ES, and the GA) fail. Additionally, the Deep GA is faster than ES, A3C, and DQN (it can train Atari in ~4 hours on one desktop or ~1 hour distributed on 720 cores), and enables a state-of-the-art, up to 10,000-fold compact encoding technique.

Subjects: **Neural and Evolutionary Computing (cs.NE)**; Machine Learning (cs.LG)

Cite as: arXiv:1712.06567 **[cs.NE]**

(or arXiv:1712.06567v3 **[cs.NE]** for this version)

Access Paper:

- Download PDF
- Other Formats

view license

Current browse context:
cs.NE

< prev | next >

new | recent | 1712

Change to browse by:

cs

cs.LG

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

6 blog links (what is this?)

DBLP - CS Bibliography

listing | bibtex

Felipe Petroski Such
Vashisht Madhavan
Edoardo Conti
Joel Lehman
Kenneth O. Stanley
...

Sometimes you don't need a computer [BONUS – NOT ON EXAM]

Slime Mold

Physarum approximates road networks in UK

Andy Adamatzky
679 subscribers

Subscribe

16 Share Save

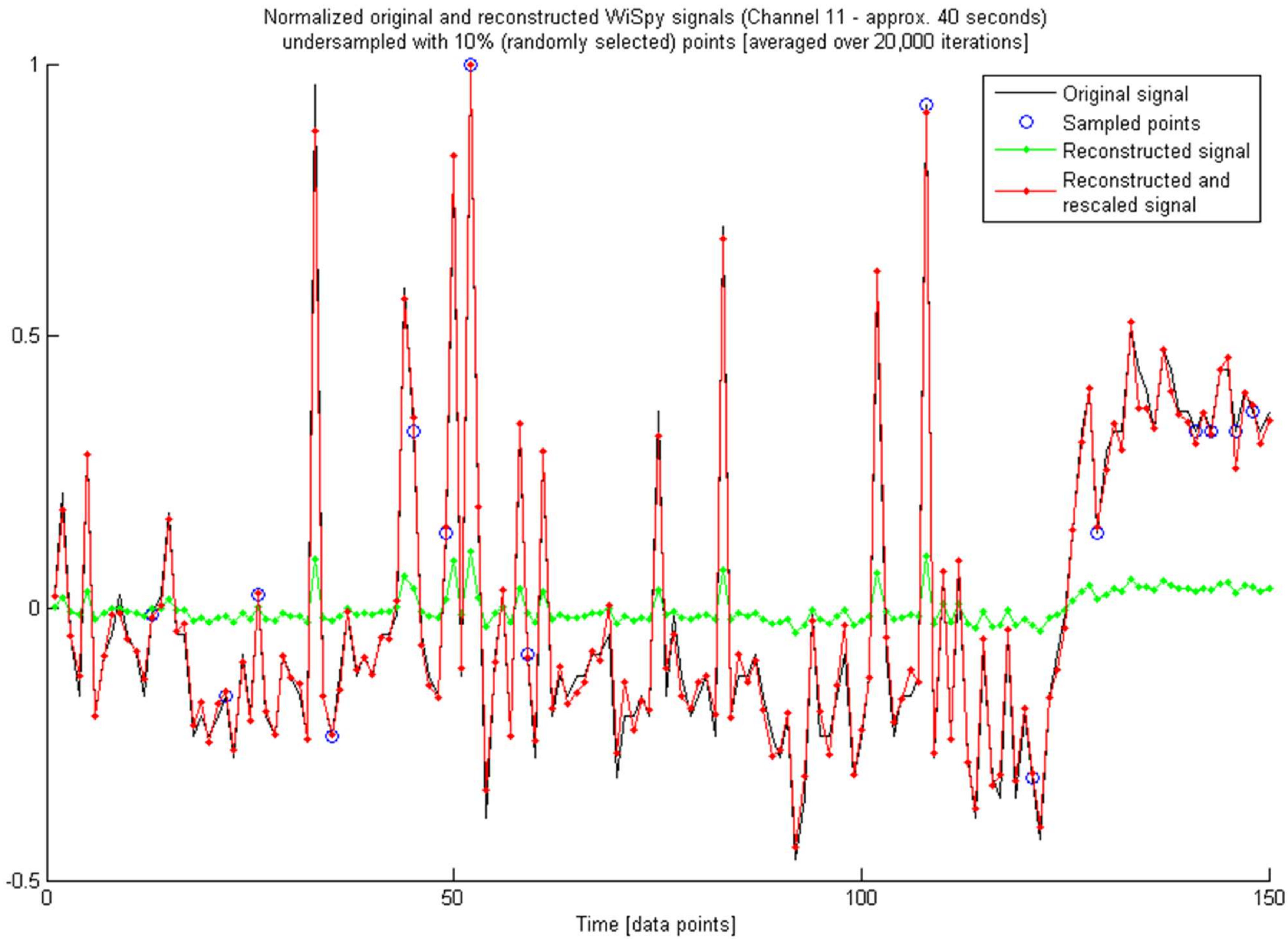
Source: https://www.youtube.com/watch?v=_DB-RAgAlVI

Slime Mold / Fungal Intelligence

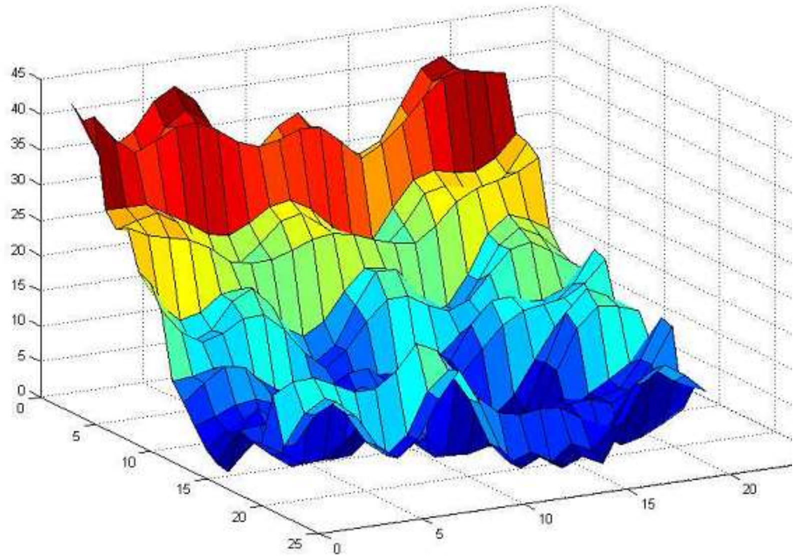
Source: https://www.youtube.com/watch?v=RVe94qa1ar4

Difficult Environments: Compressive Sensing [BONUS – NOT ON EXAM]

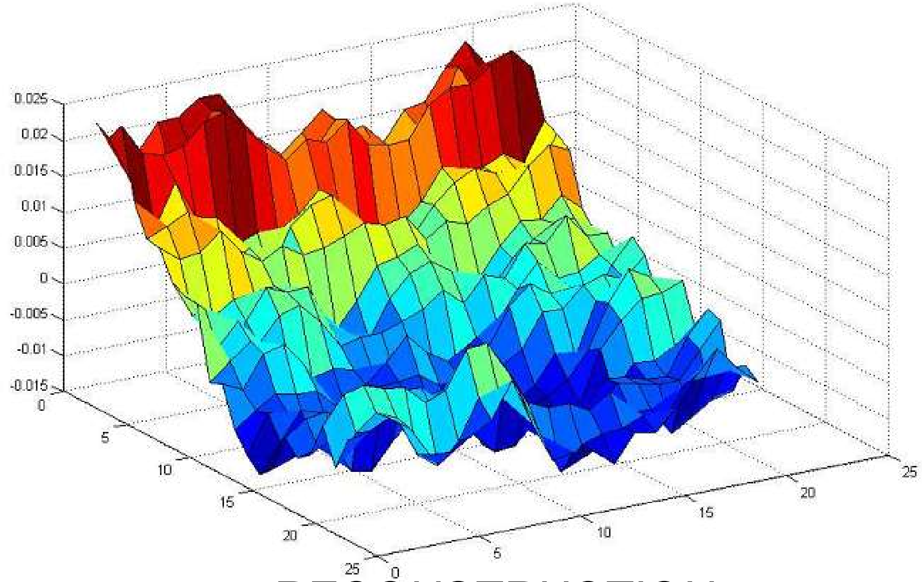
Compressive Sensing



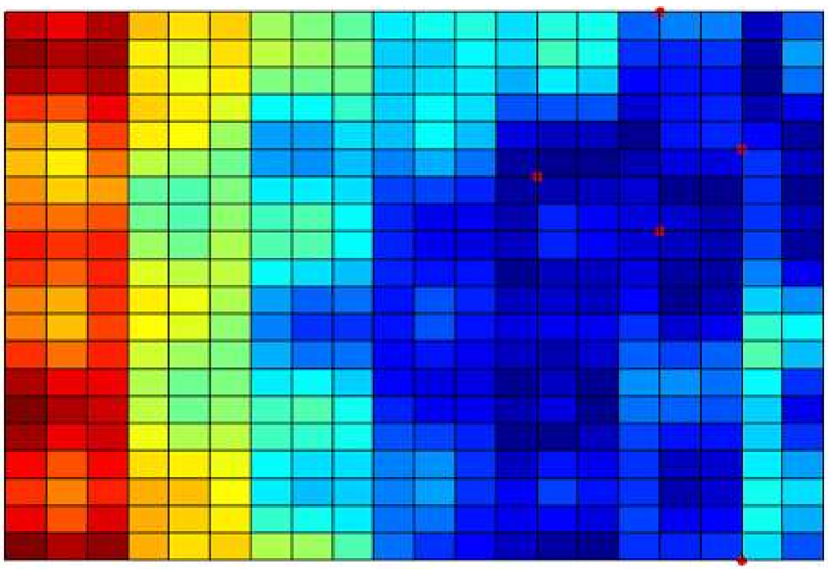
Compressive Sensing



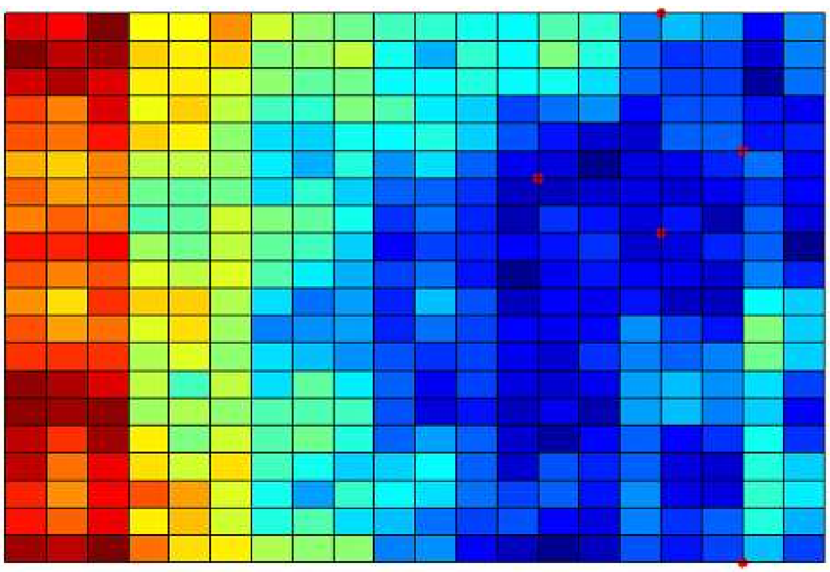
ORIGINAL



RECONSTRUCTION



ORIGINAL



RECONSTRUCTION

Fuzzy Logic

Linguistic Rules

Imagine loan application processing rules

```
If credit score good then risk is low
If credit score bad then risk is high
If credit score medium then risk is average
```

What does it mean: low, high, good, bad?

Fuzzy Logic: the Idea

- **Boolean ("crisp") logic**

true **false**

- **Fuzzy (many valued) logic**

true **false**

Fuzzy Logic: the Idea

- **Boolean (“crisp”) logic**

cold hot

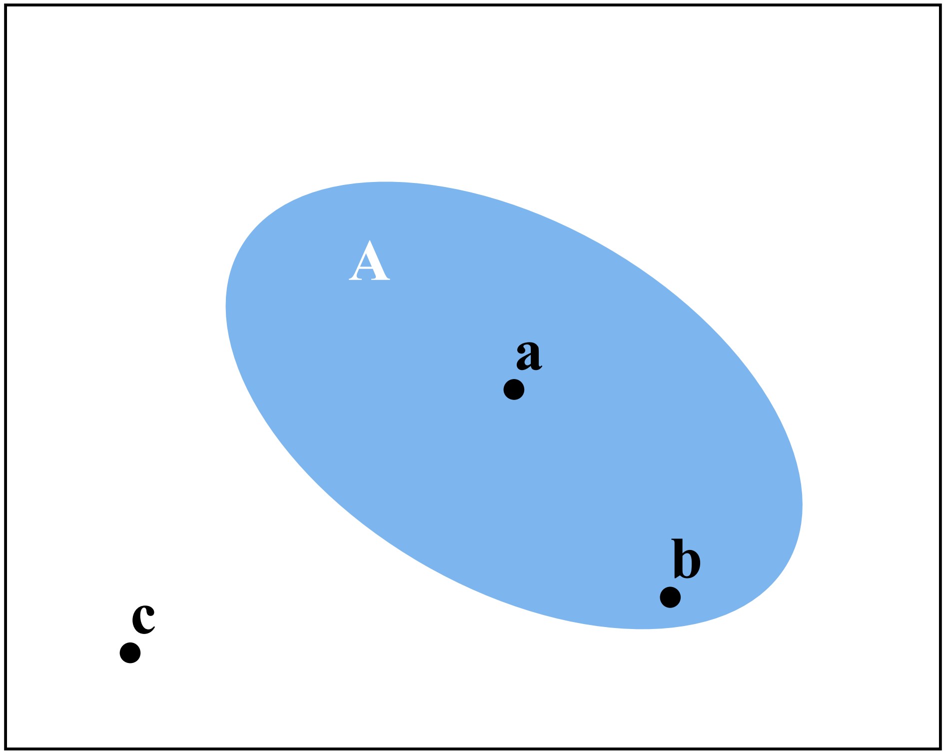
- **Fuzzy (many valued) logic**

cold warm hot

Fuzzy Logic: Fuzzy Sets

"Crisp" Set A

an element is a set member or not



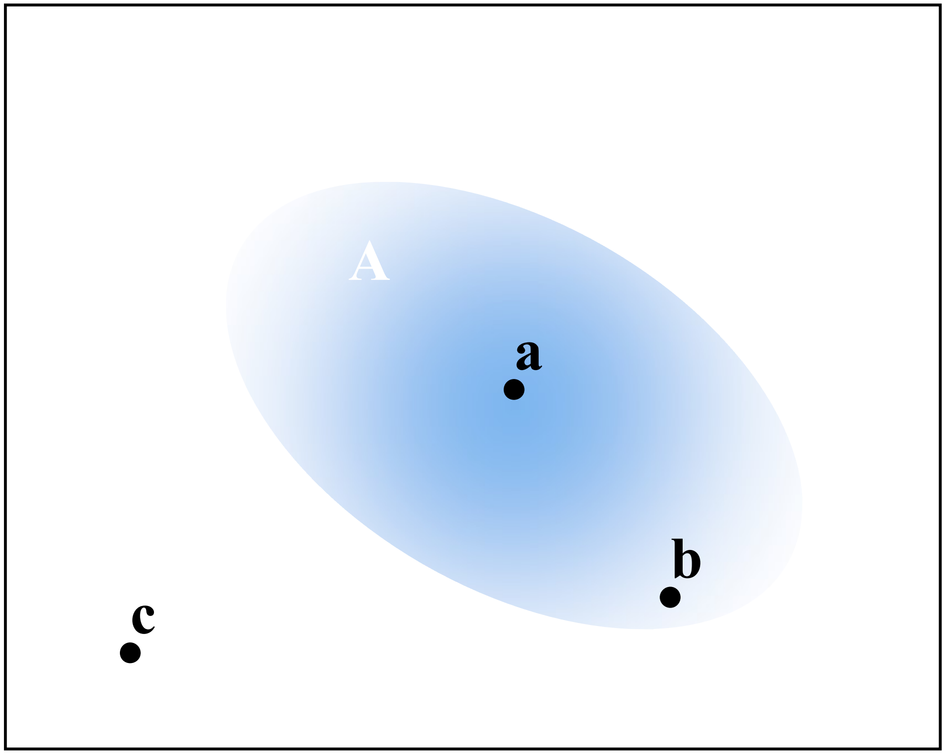
$a \in A$

$b \in A$

$c \notin A$

Fuzzy Set A:

an element is a set member
with some membership degree μ

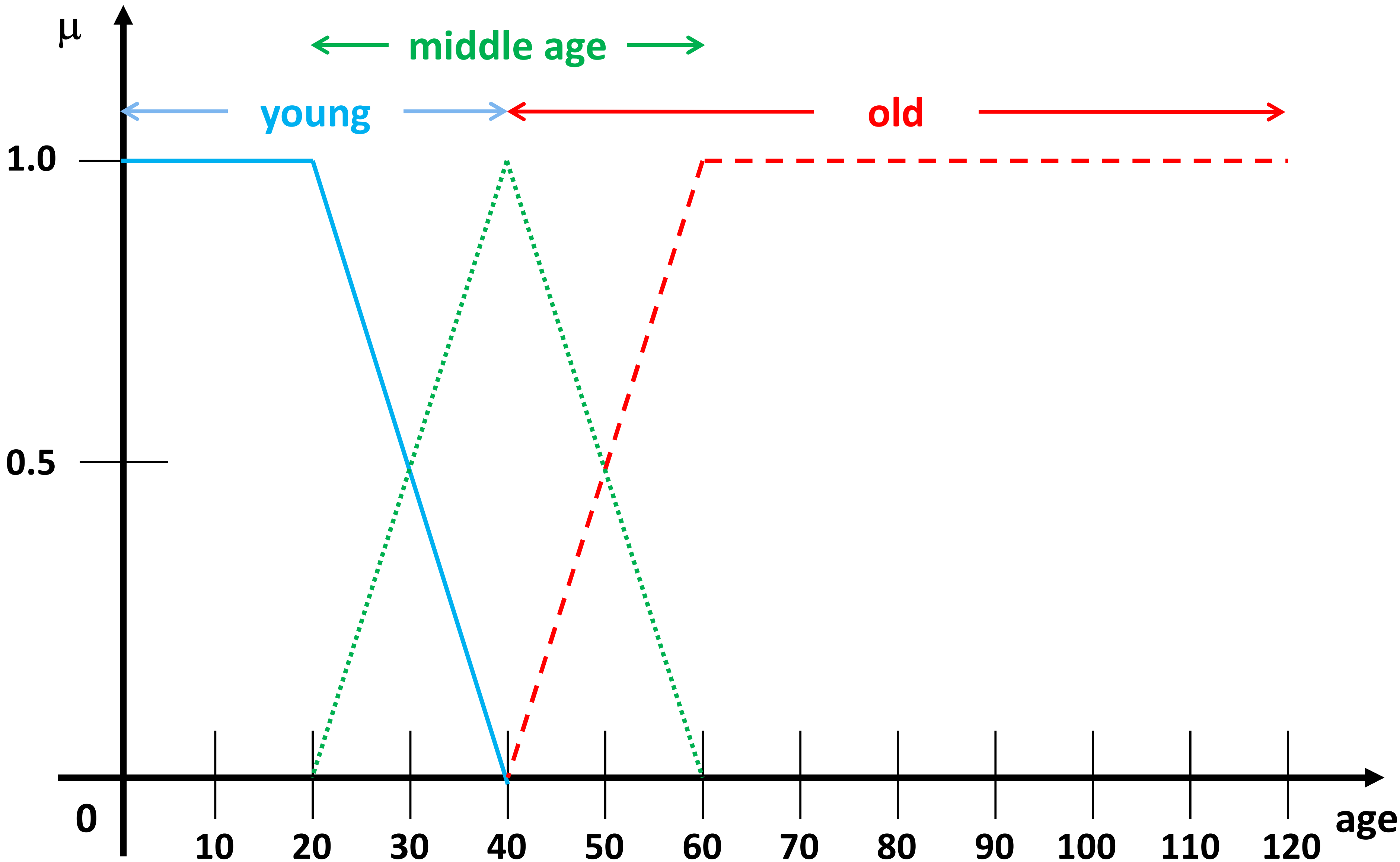


$\mu(a) = 1.0$

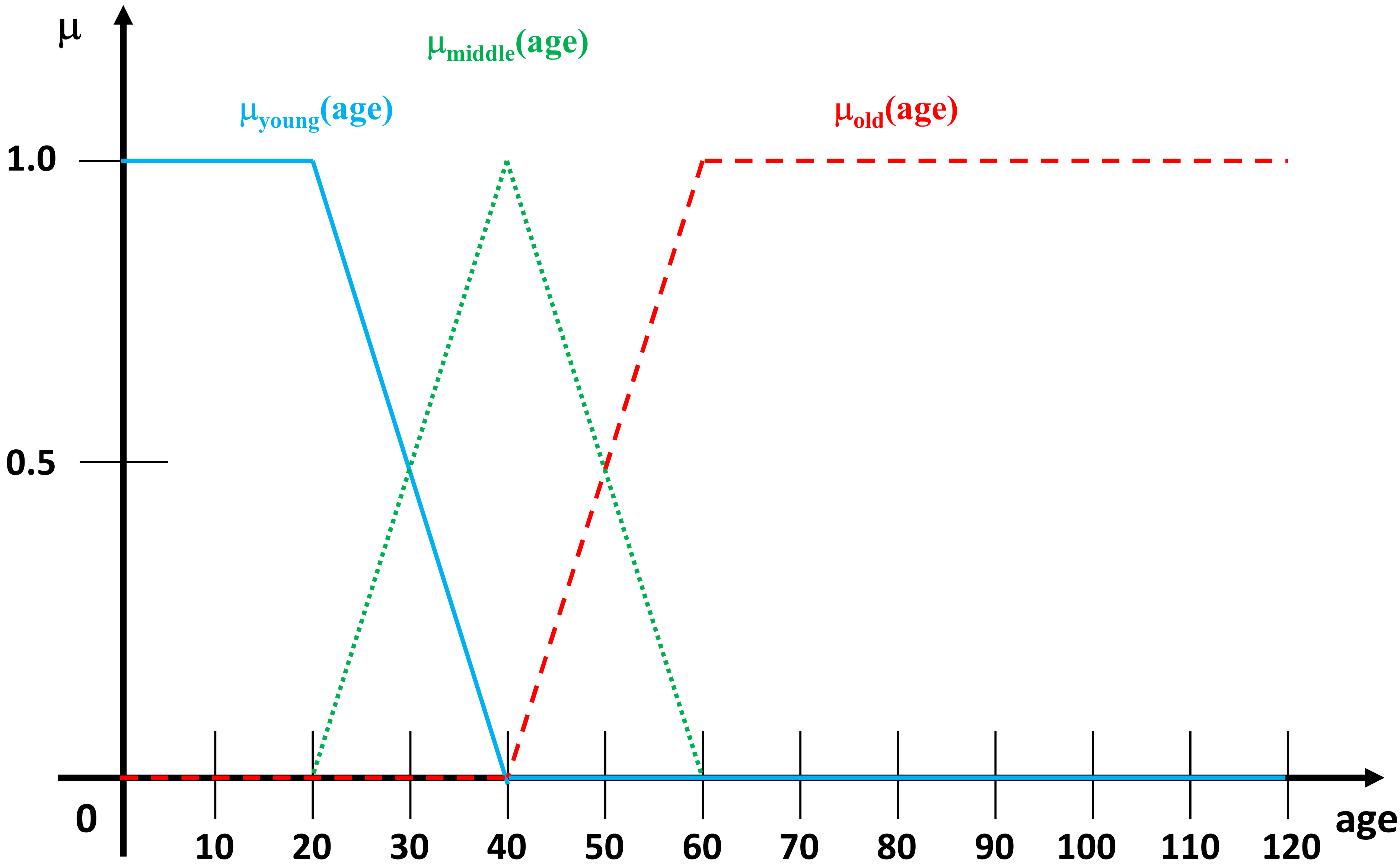
$\mu(b) = 0.1$

$\mu(c) = 0.0$

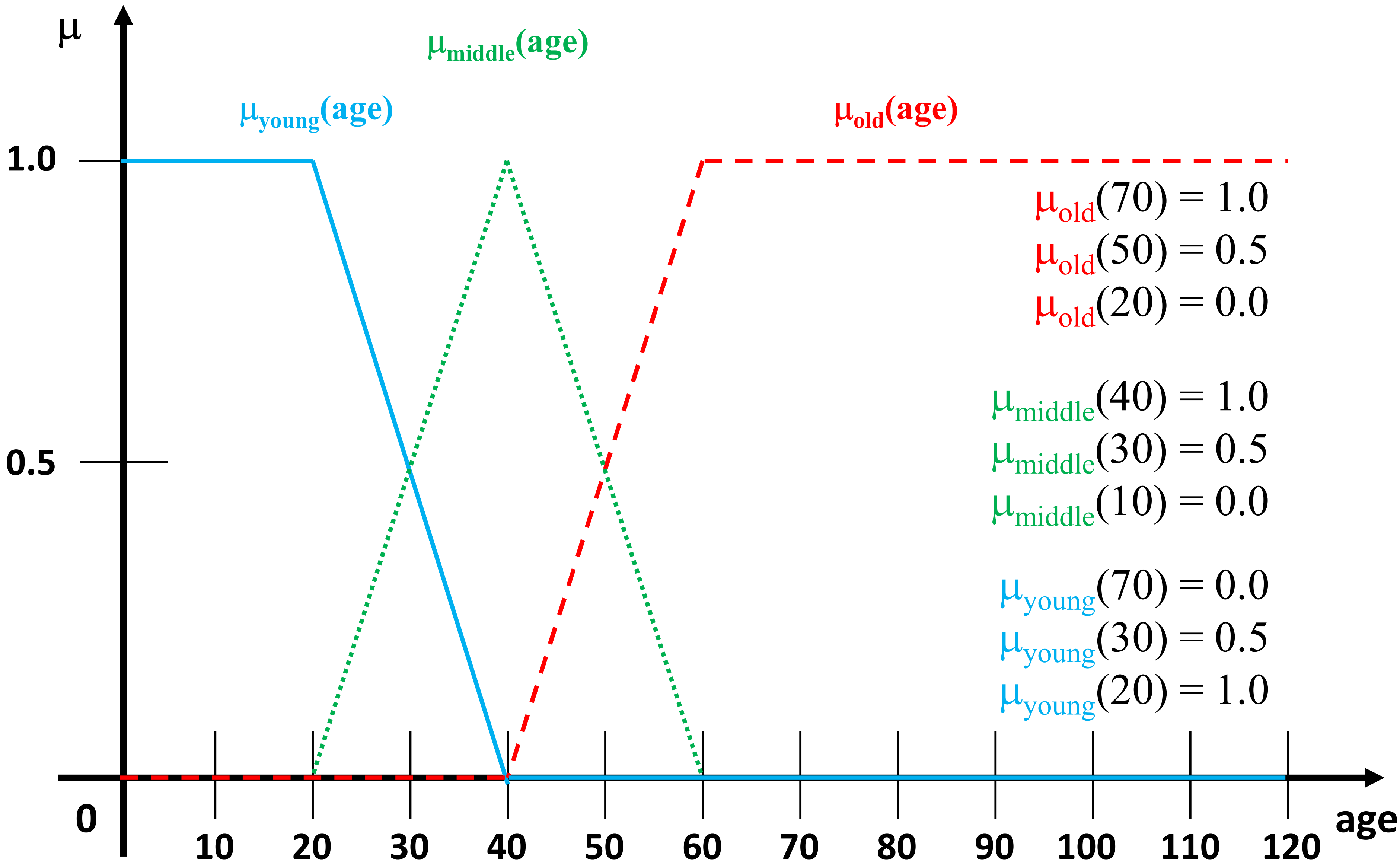
Fuzzy Logic: Fuzzy Sets



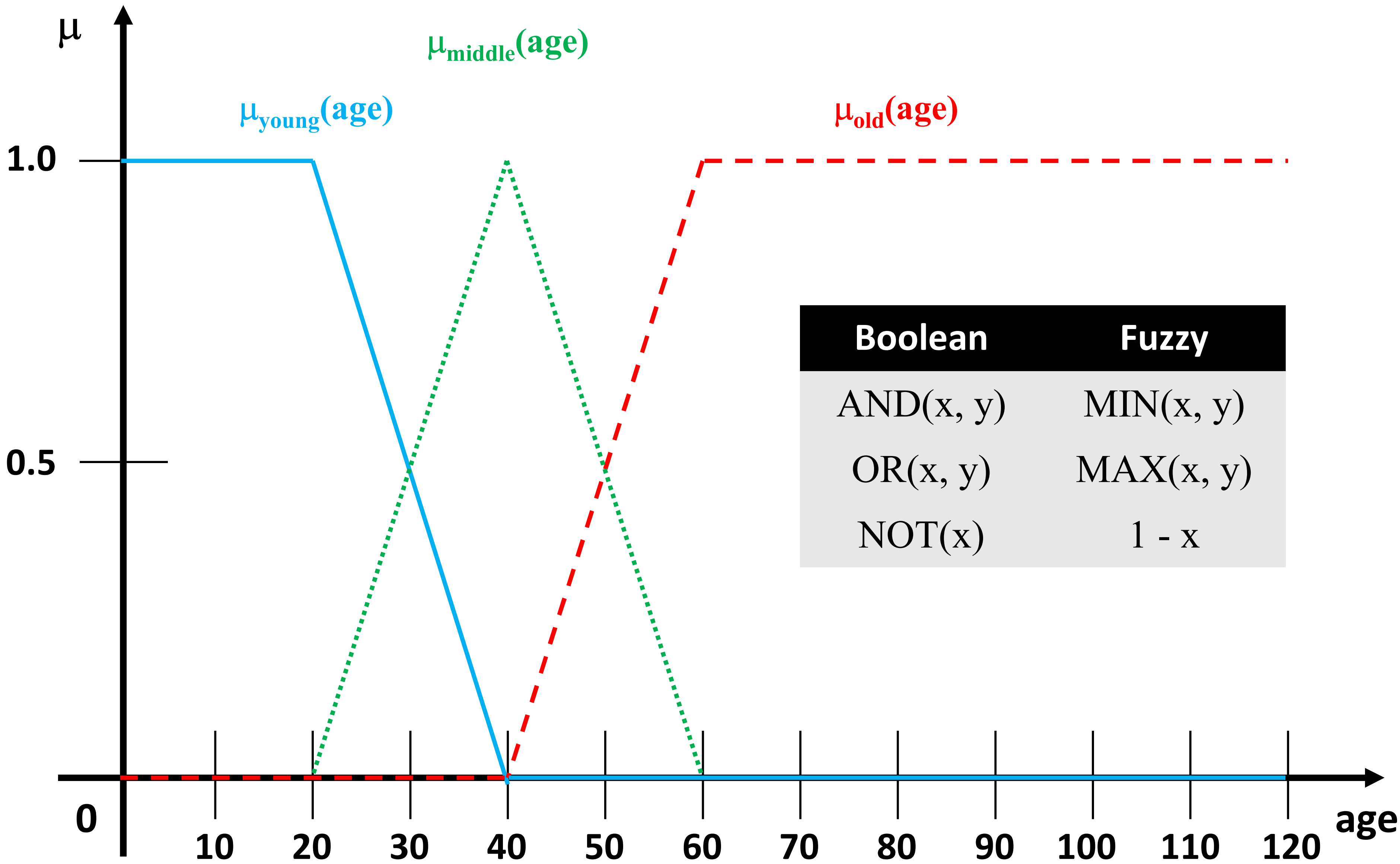
Fuzzy Logic: Membership Functions



Fuzzy Logic: Membership Functions

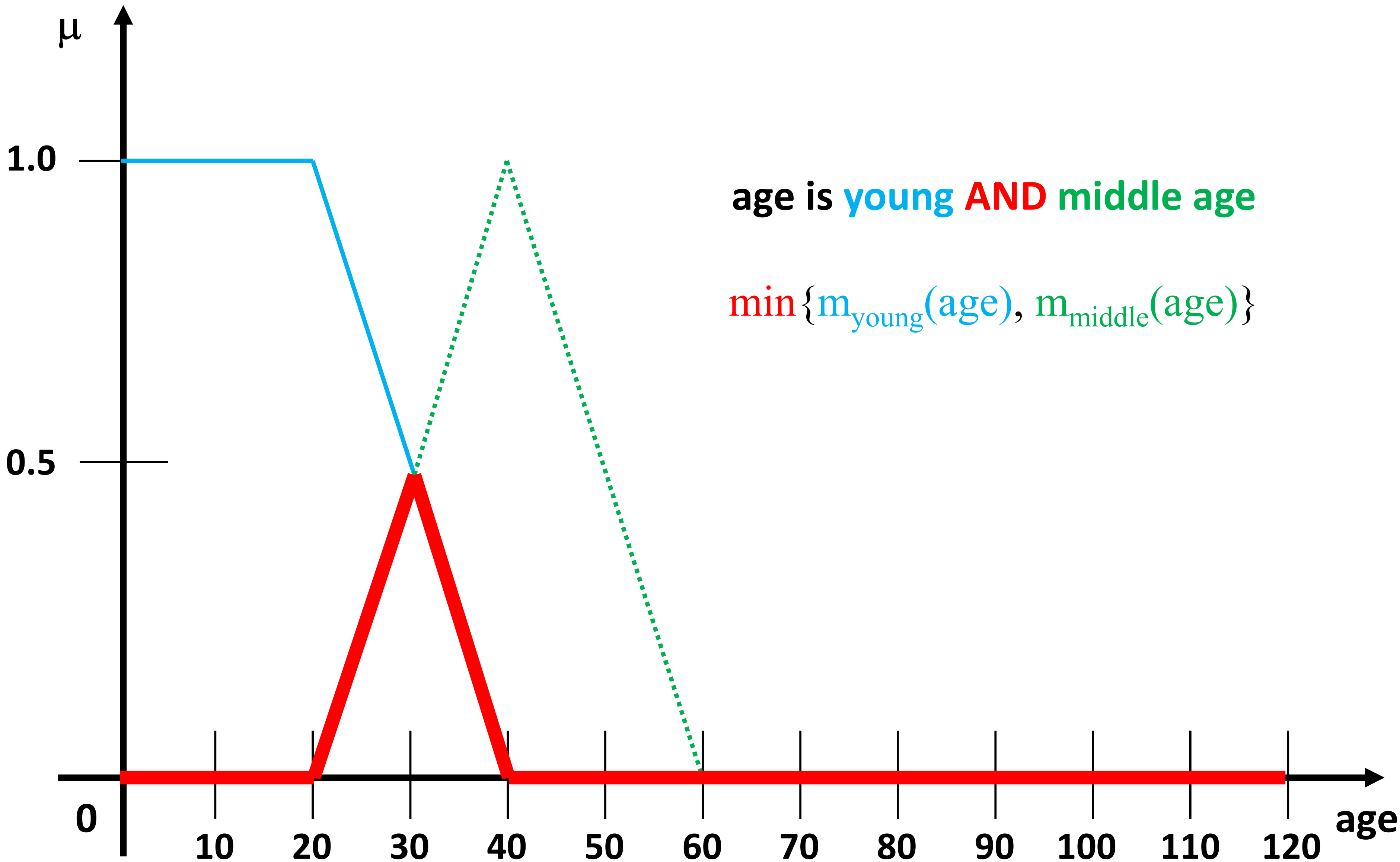


Fuzzy Logic: Logic Operators

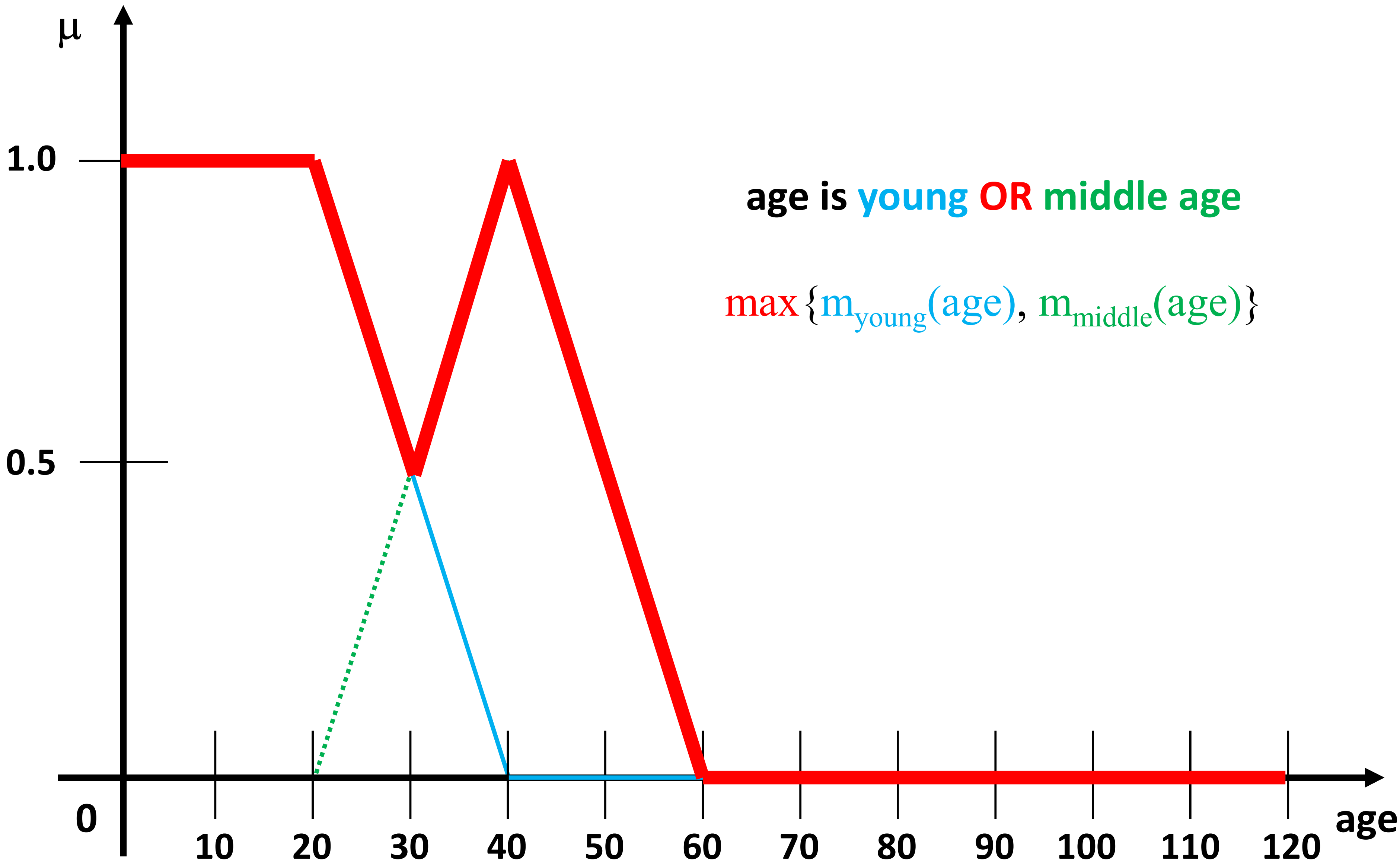


Boolean	Fuzzy
AND(x, y)	MIN(x, y)
OR(x, y)	MAX(x, y)
NOT(x)	1 - x

Fuzzy Logic: the AND Operator



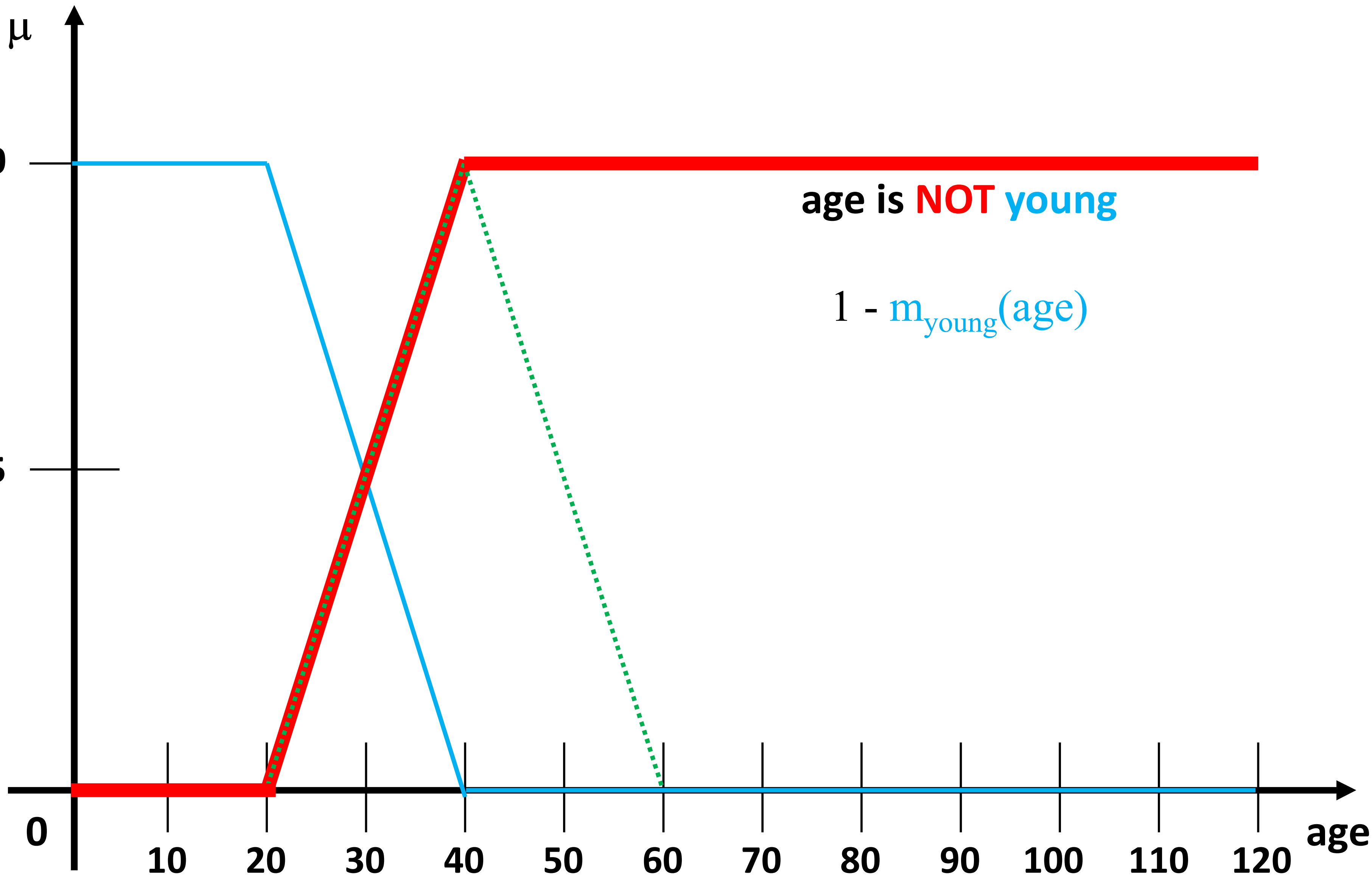
Fuzzy Logic: the OR Operator



age is young OR middle age

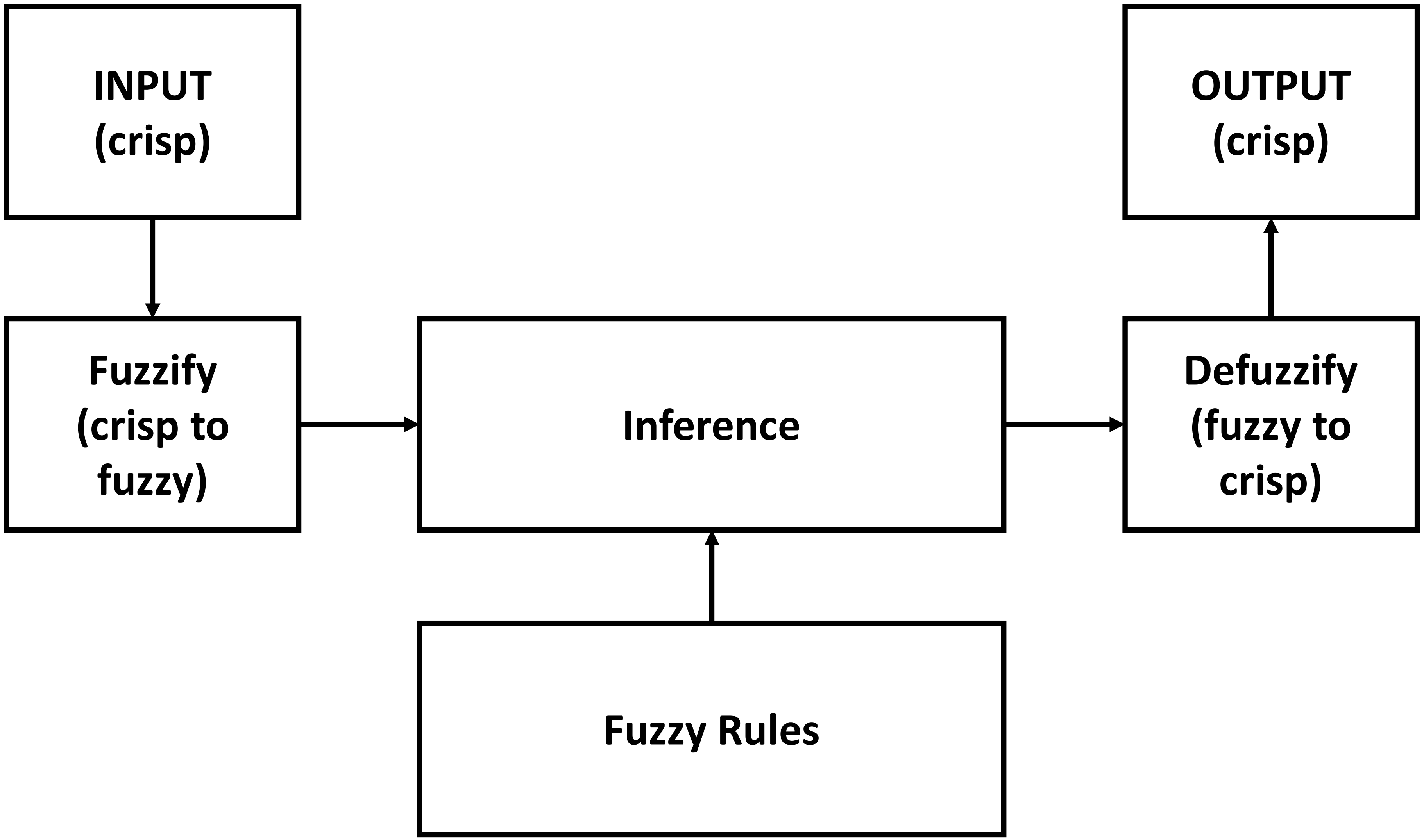
$$\max\{m_{young}(age), m_{middle}(age)\}$$

Fuzzy Logic: the NOT Operator

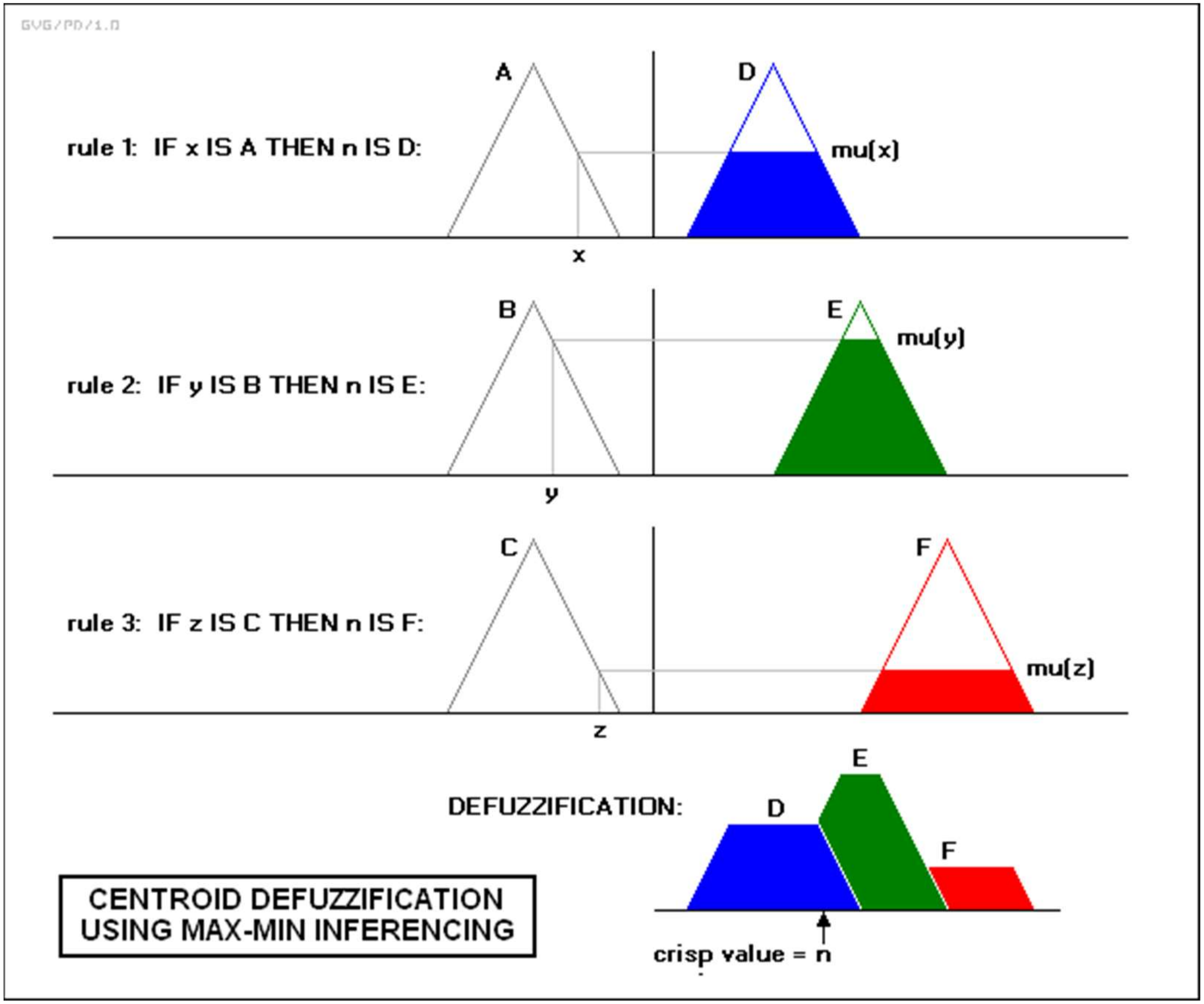


Fuzzy Inference Systems

Fuzzy Inference System



Defuzzification



Source: https://en.wikipedia.org/wiki/Defuzzification

Fuzzy Inference Systems

- Simple method for applying machine learning
- Replicates the logical process of human reasoning
- Fuzzy logic in AI considers inference as a method of spreading elastic restrictions.
- Enables the construction of nonlinear functions with any degree of complexity.
- It may be configured to fail safely if any feedback sensor fails or gets damaged
- Modifiable to enhance or raise system performance.
- Fuzzy logic can control nonlinear systems that might be challenging to handle mathematically.

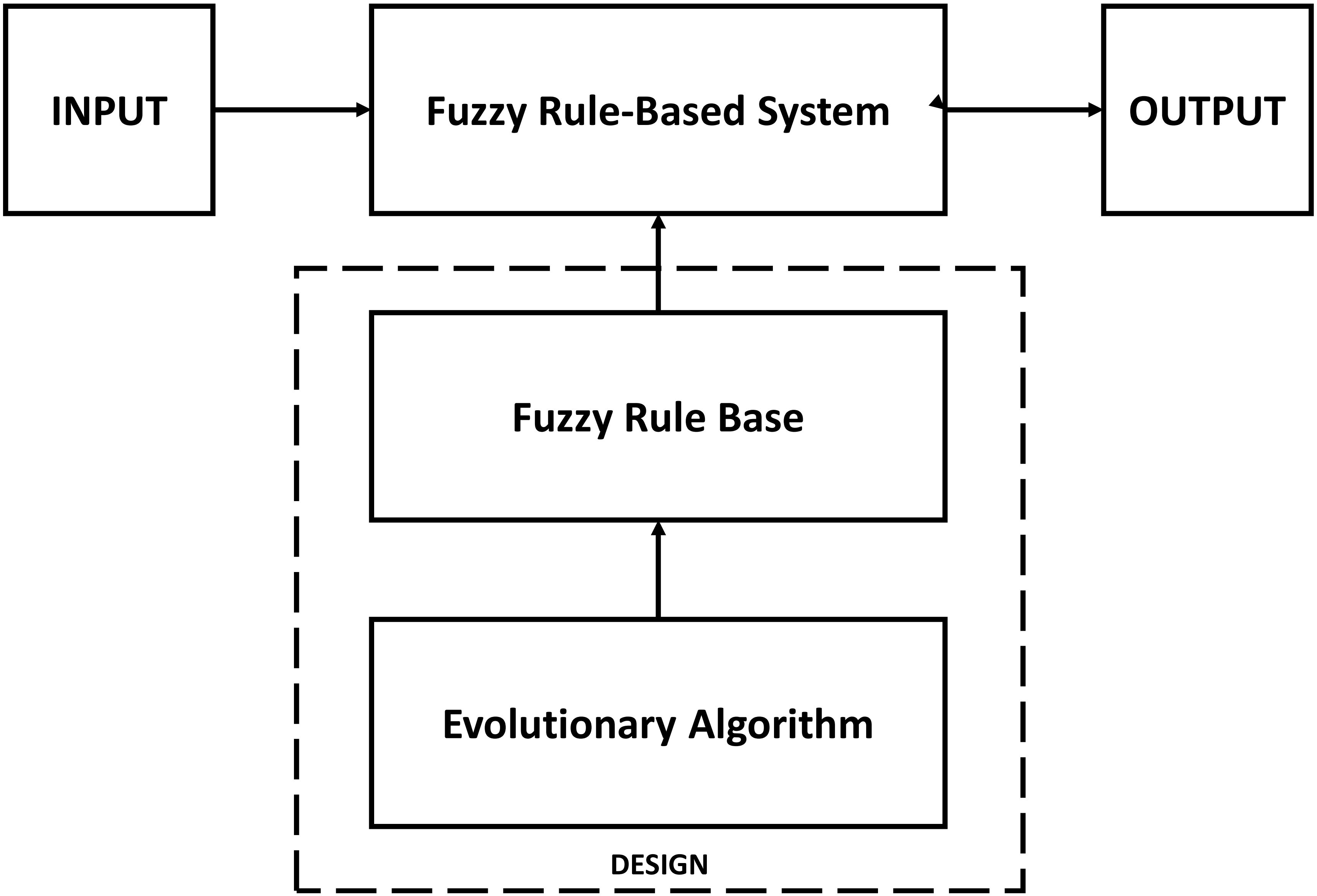
Genetic Fuzzy Systems

Genetic Fuzzy Systems

Genetic fuzzy systems are fuzzy systems constructed using genetic algorithms or genetic programming (the process of evolution is applied to identify fuzzy system structure and parameters).

Goal: generation of fuzzy rules from a given input-output data set

Genetic Fuzzy Systems



Coordinated / Cooperative Population-based Algorithms

Swarm Optimization

Flocking and Schooling

Source: https://www.youtube.com/watch?v=15B8qN9dre4

Murmuration

Source: https://www.youtube.com/watch?v=V4f_1_r80RY

Swarm Intelligence [Wikipedia]

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence.

SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment.

The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents.

Emergence [Wikipedia]

In philosophy, systems theory, science, and art, emergence occurs when a complex entity has properties or behaviors that its parts do not have on their own, and emerge only when they interact in a wider whole.

Emergence plays a central role in theories of integrative levels and of complex systems. For instance, the phenomenon of life as studied in biology is an emergent property of chemistry and quantum physics.

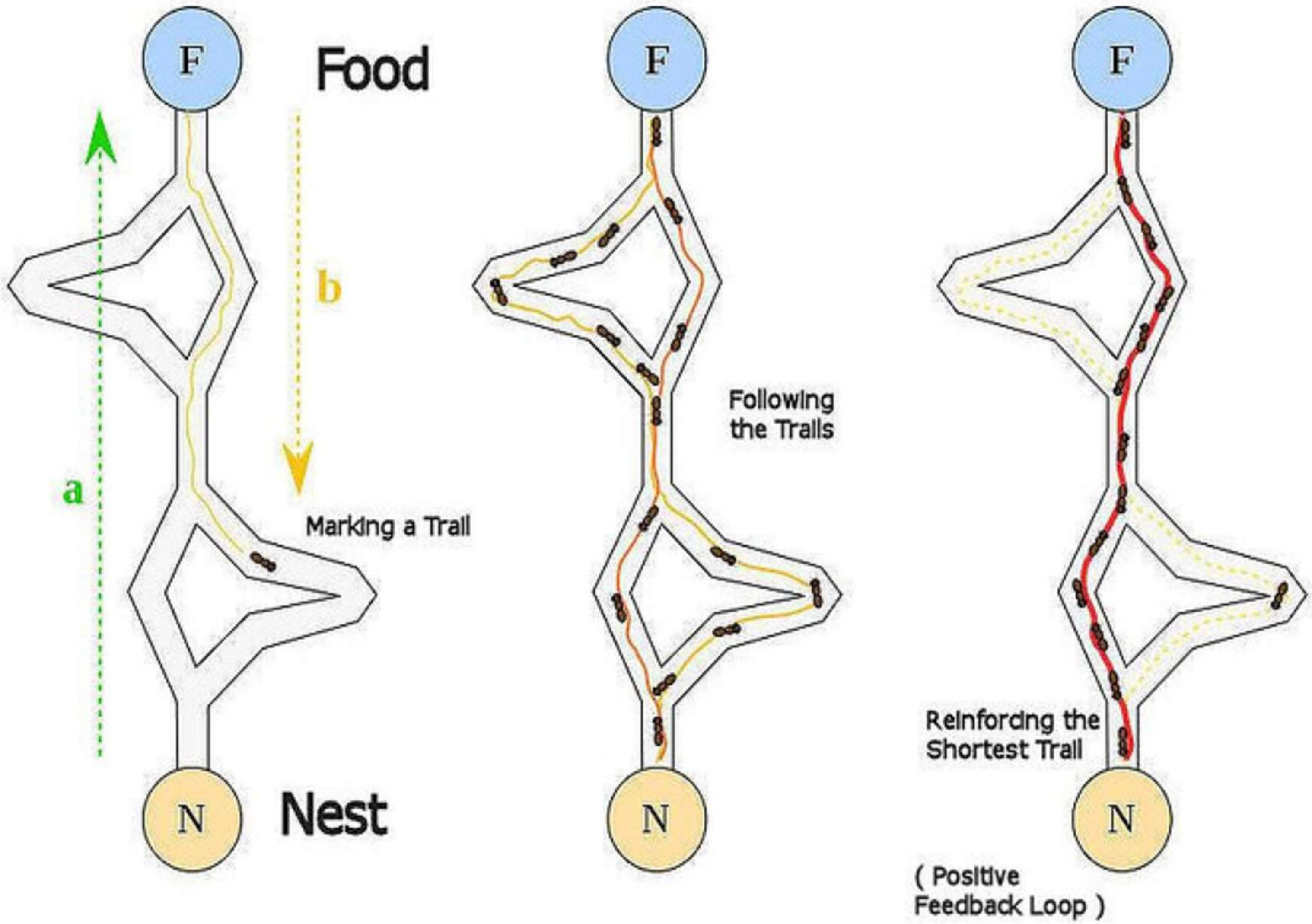
Ant Colony Optimization

Stigmergy

Stigmergy (coined by French biologist Pierre-Paul Grasse) = interaction through the environment

Two individuals interact <u>indirectly</u> when one of them modifies the environment and the other responds to the new environment at a later time

Ant Colony Optimization: The Idea

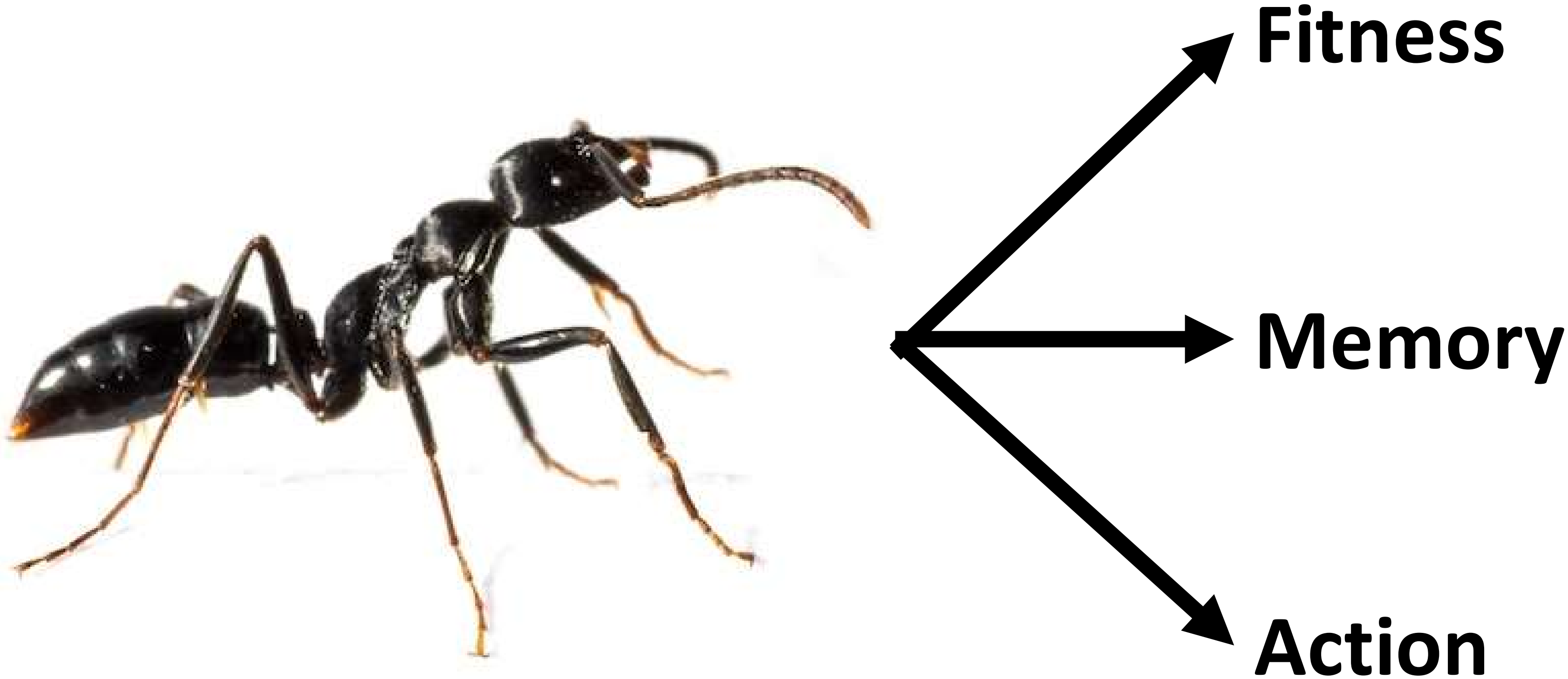


Source: https://wikipedia.org/

Pheromone Trail

- **Individual ants deposit pheromones while travelling between nest and food source**
- **Pheromone trail gradually evaporates over time**
- **Pheromone trail strength accumulated with multiple ants using path.**

Artificial Ant Properties

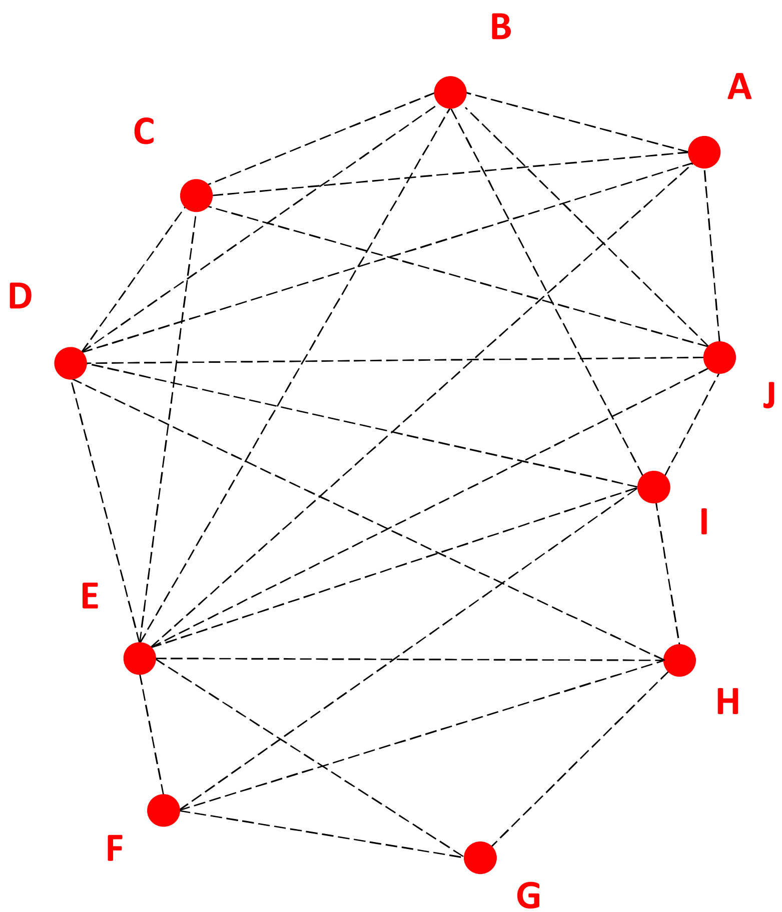


Source: https://www.freepik.com/free-photo/close-up-black-ant_903352.htm Image by onlyyouqj on Freepik

Artificial Ant Properties

- **Memory: the list of visited places (nodes in the graph)**
- **Best fitness: the lowest cost total distance traveled between places**
- **Action: select next location (node) to visit (and leave pheromones along the way)**

Traveling Salesman Problem



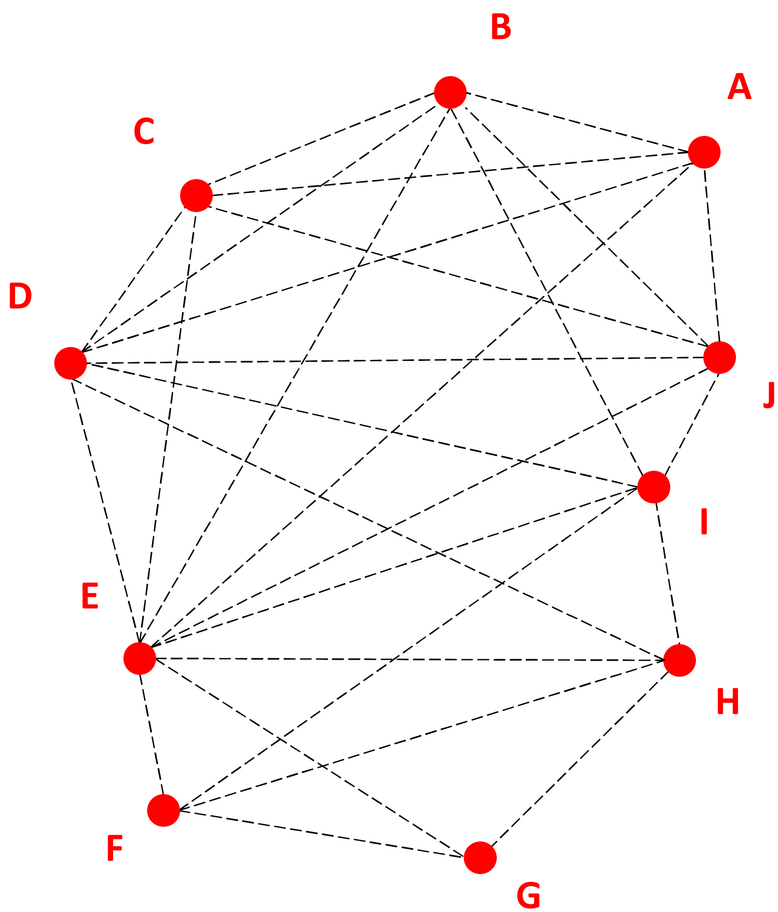
Problem:

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once.

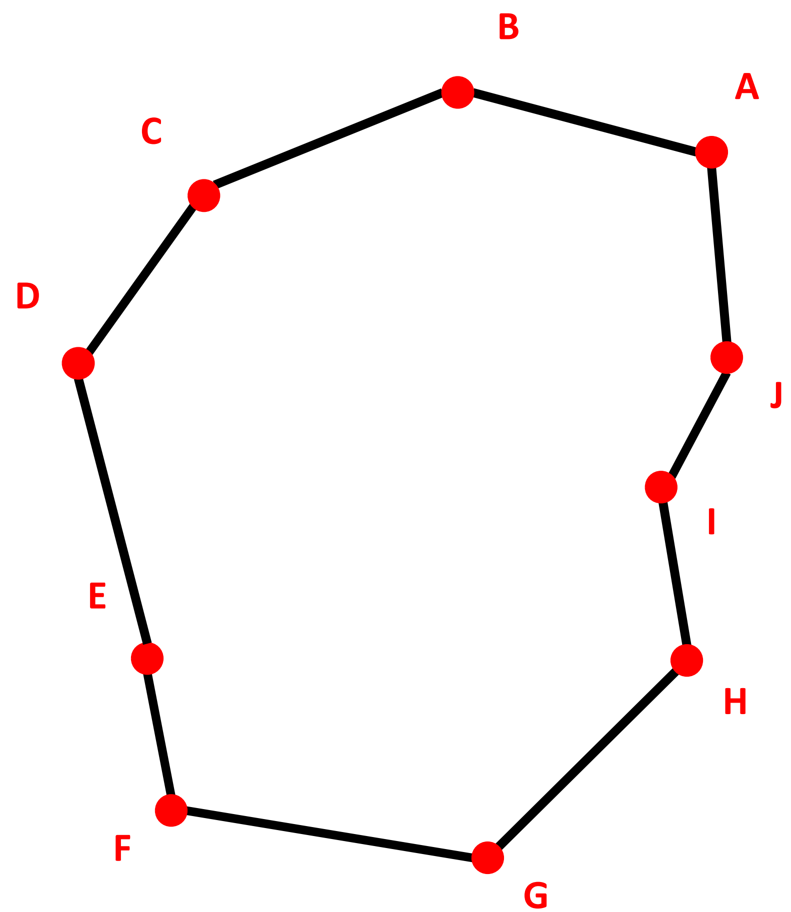
Solution:

Shortest possible path/route such that he visits each city exactly once and returns to the origin city.

Traveling Salesman Problem



PROBLEM



SOLUTION

Artificial Ant Properties

Path Cost

I was there…

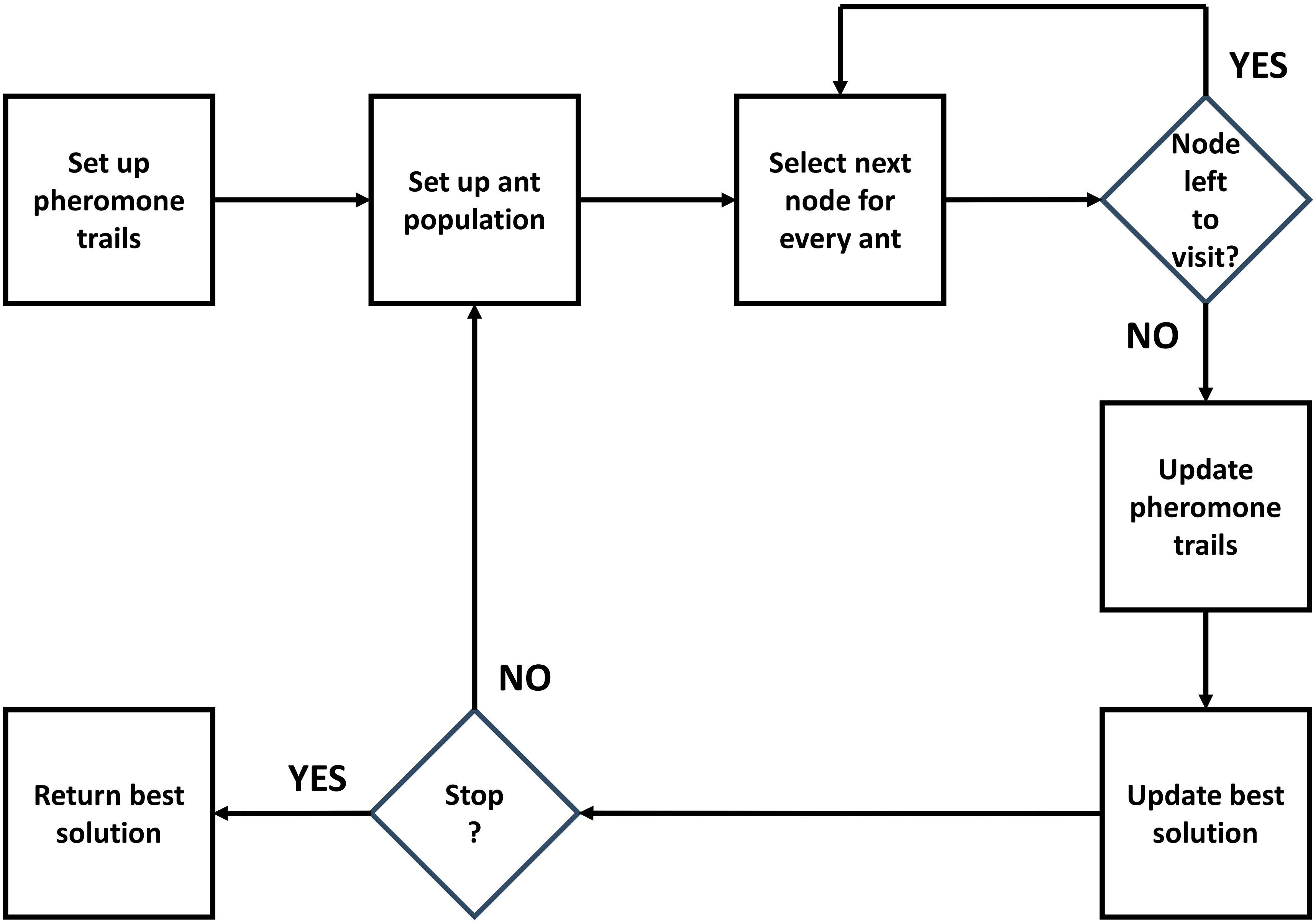
Next?

Source: https://www.freepik.com/free-photo/close-up-black-ant_903352.htm Image by onlyyouqj on Freepik

TSP with Ant Colony Optimization

1. Initialization (ants, pheromone trails)
2. Randomly place ants at nodes
3. Build tours / paths
4. Deposit pheromone, update trail, solution
5. Repeat or exit

TSP with Ant Colony Optimization



TSP with Ant Colony Optimization

1) Initialize the pheromone trails. Create the concept of pheromone trails between nodes, and initialize their intensity values.

2) Initialize the population of ants.

3) Select next location (node) for each ant. Repeat until each ant has visited all locations once

TSP with Ant Colony Optimization

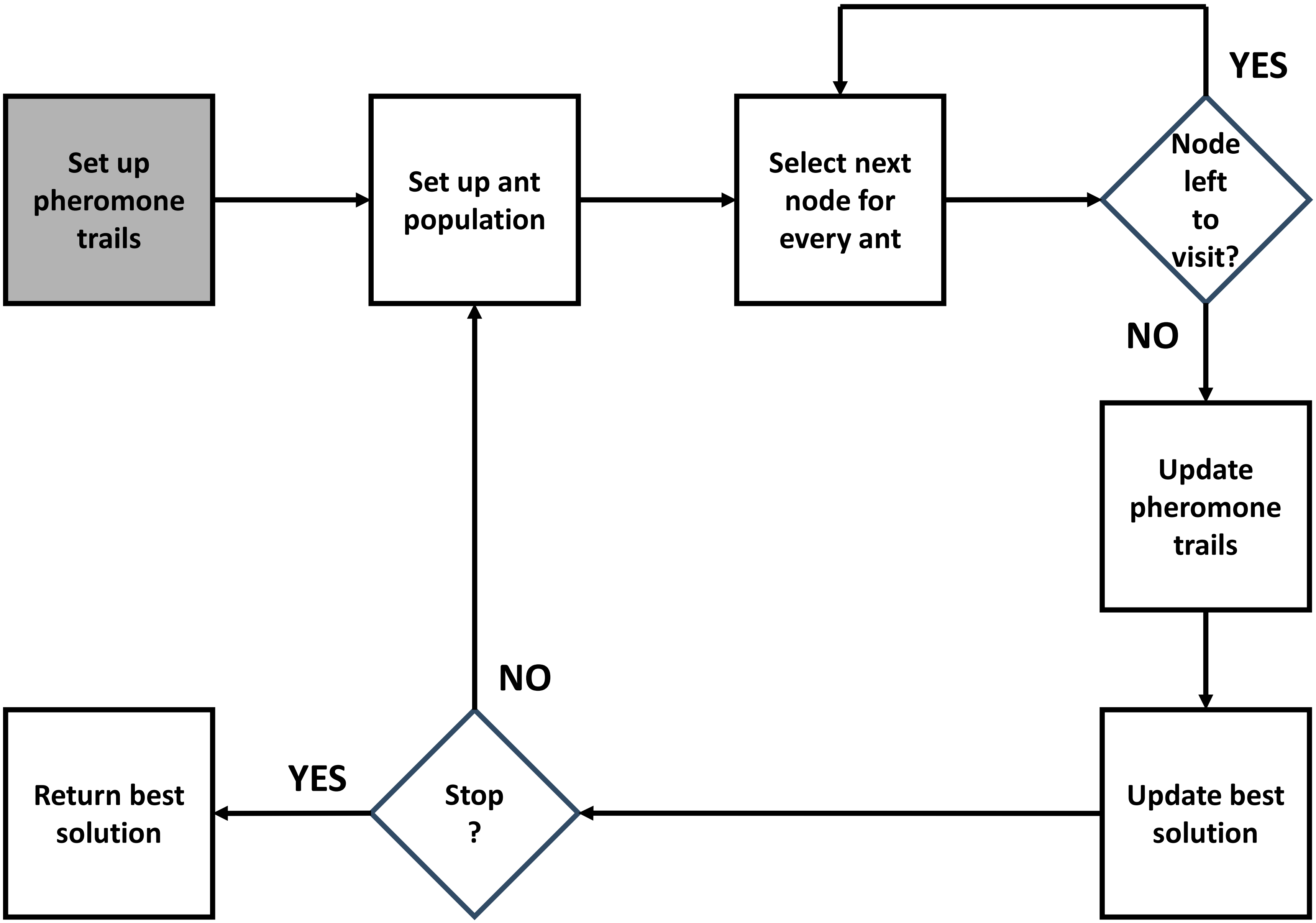
4) Update the pheromone trails/intensity (edges) based on the ant movements on them. Factor in pheromone evaporation

4) Update the best solution given the total distance covered by each ant.

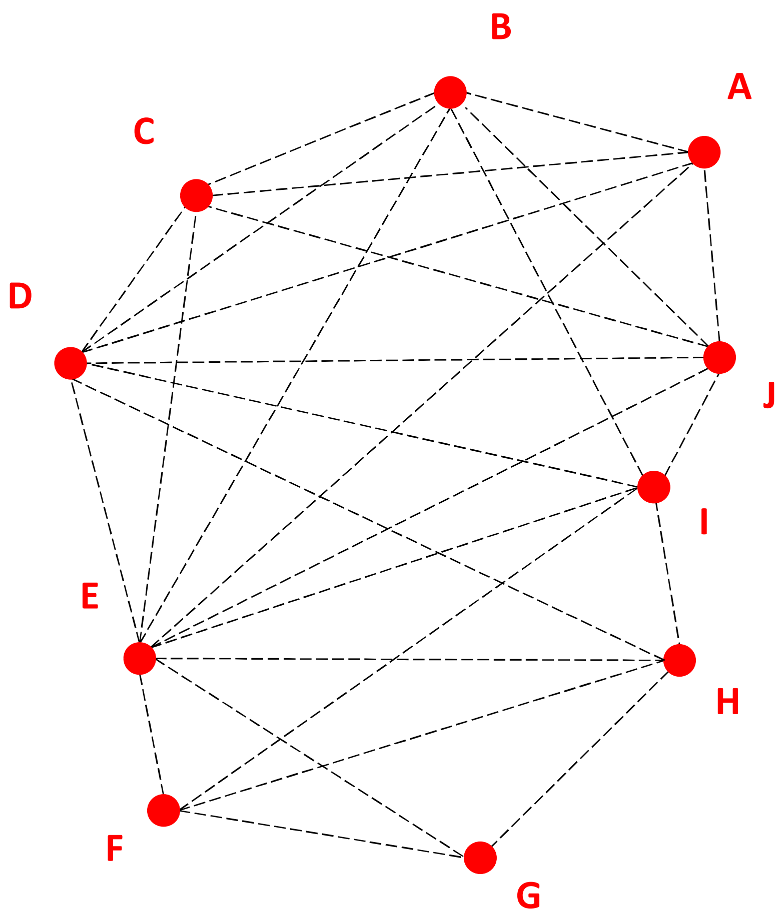
5) Set the stopping criteria. The process of ants visiting attractions repeats for several iterations.

- one iteration is every ant visiting all attractions once
- the stopping criterion determines the total number of iterations to run
- more iterations allow ants to make better decisions

TSP with Ant Colony Optimization

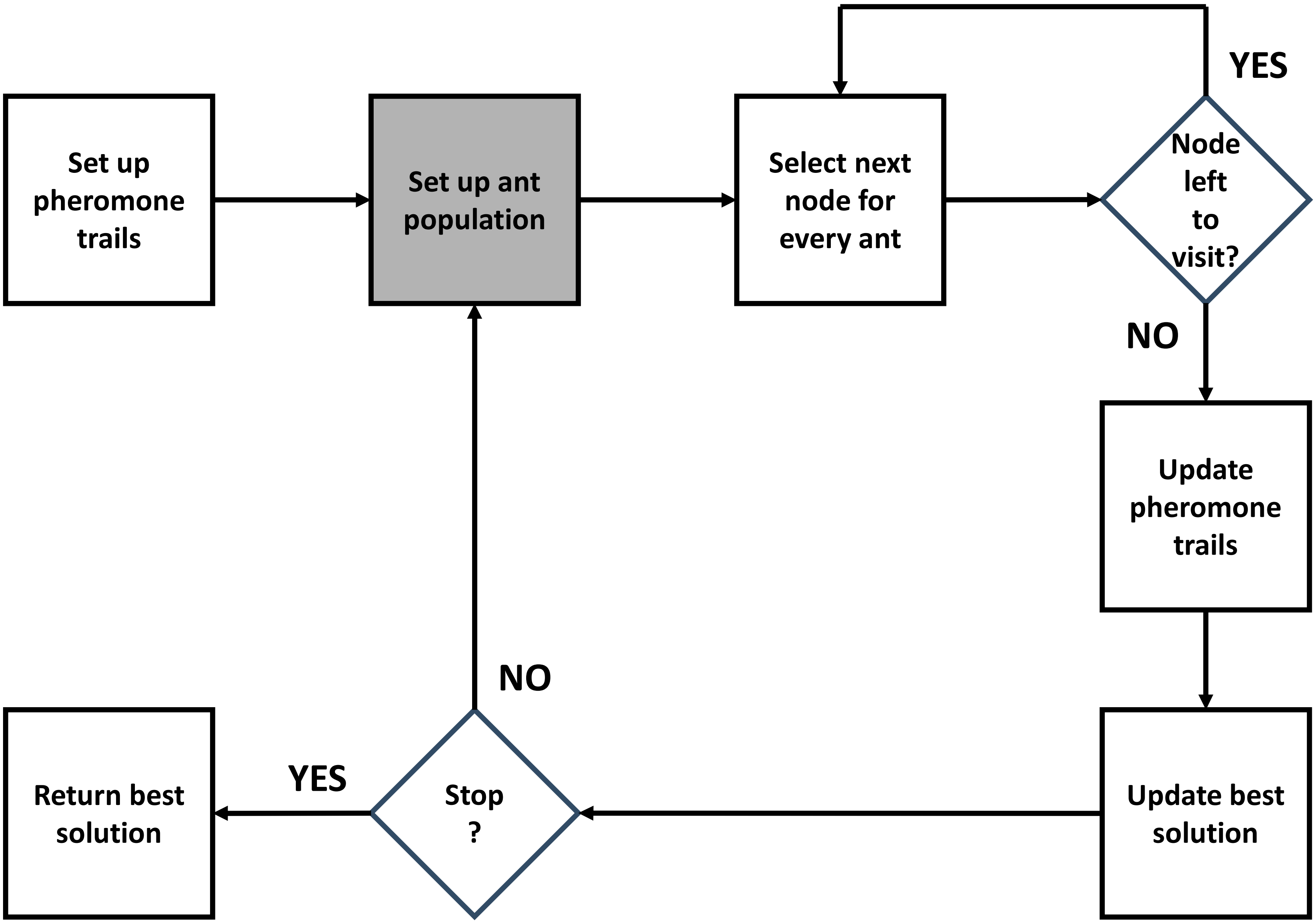


Set Up Pheromone Trails

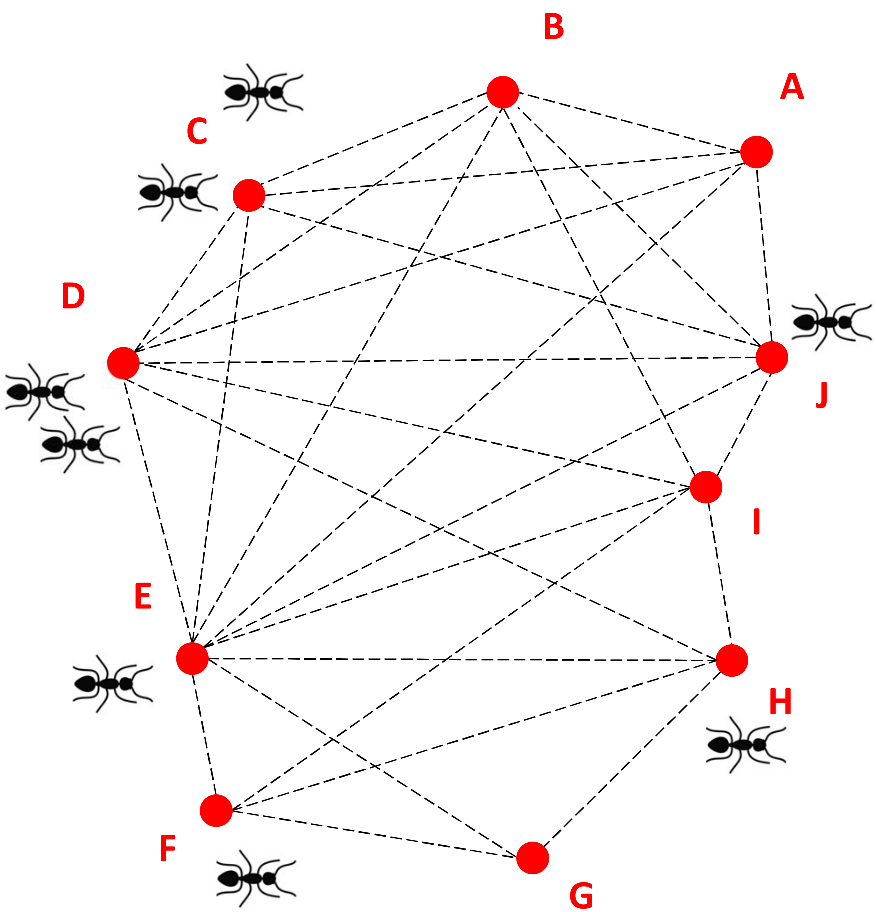


Set all pheromone trail values to 1 (for all edges)

TSP with Ant Colony Optimization



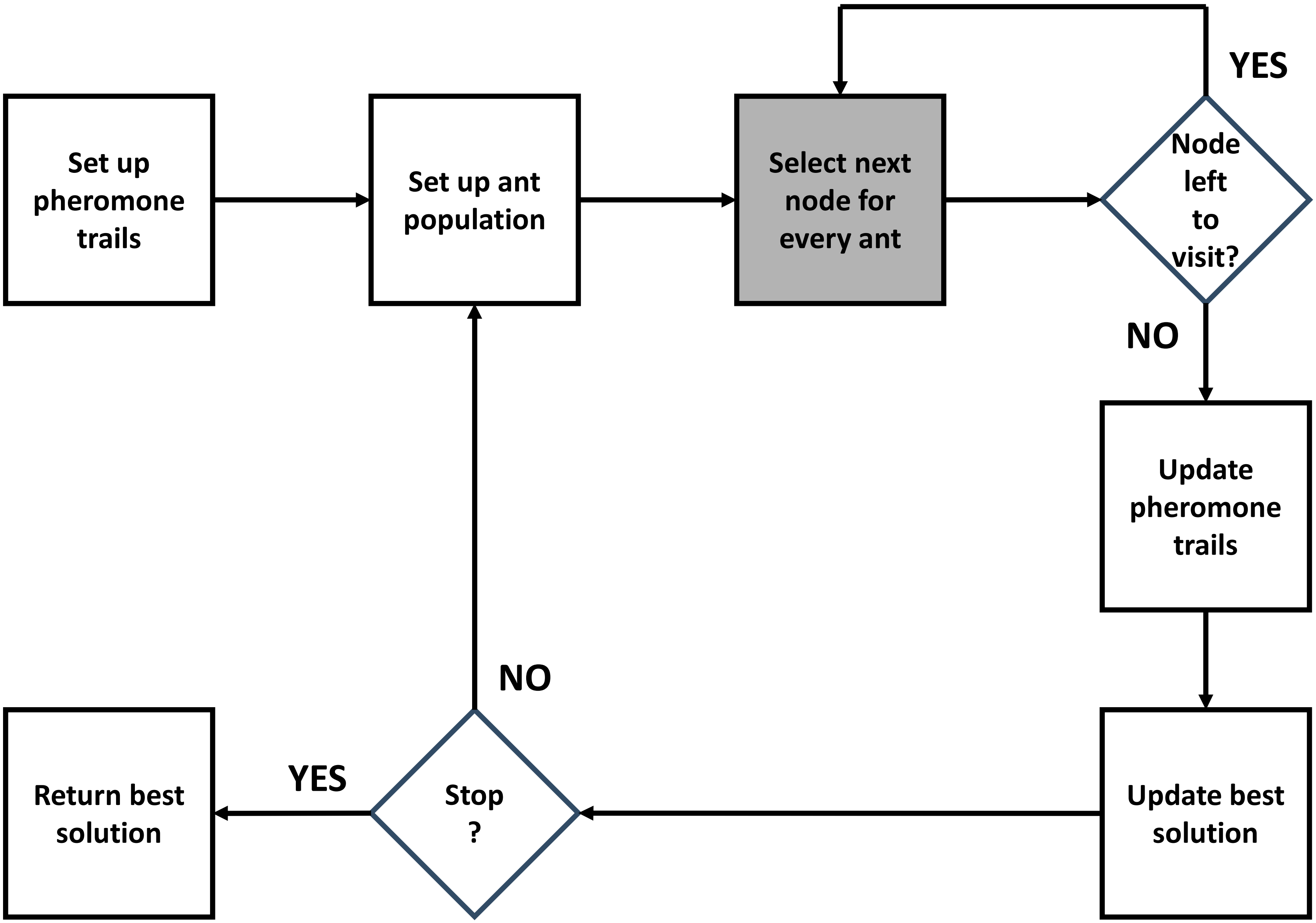
Set Up Ant Population



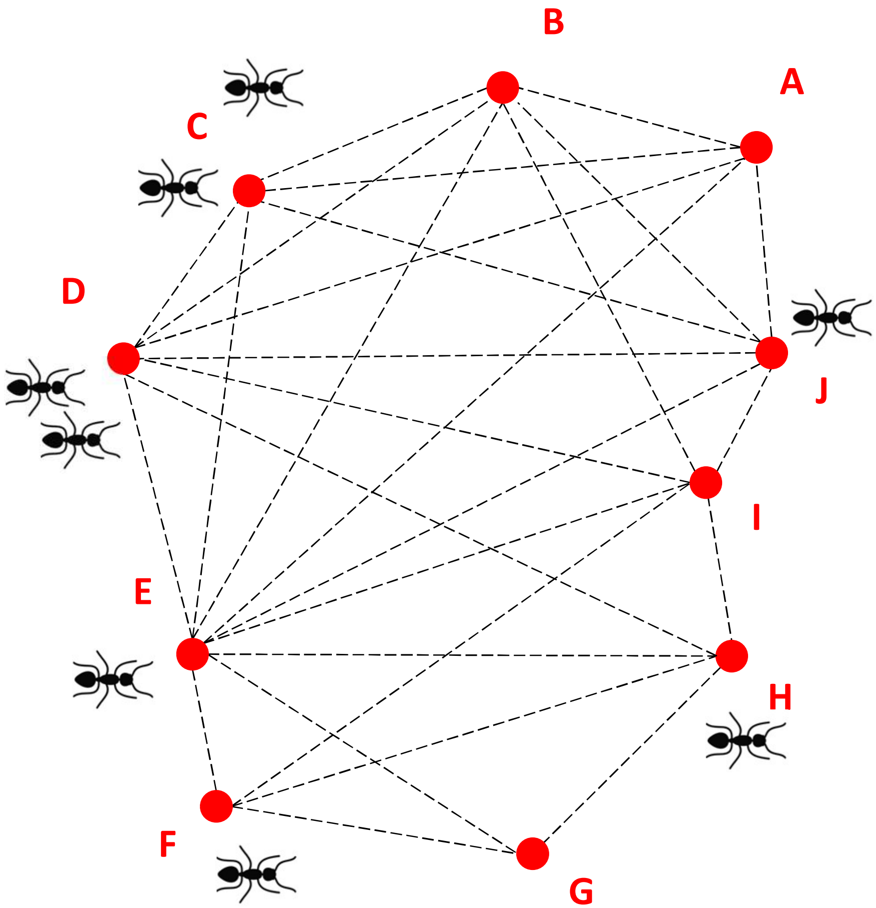
Set up the ant population:

- **randomly assign ants to nodes**

TSP with Ant Colony Optimization



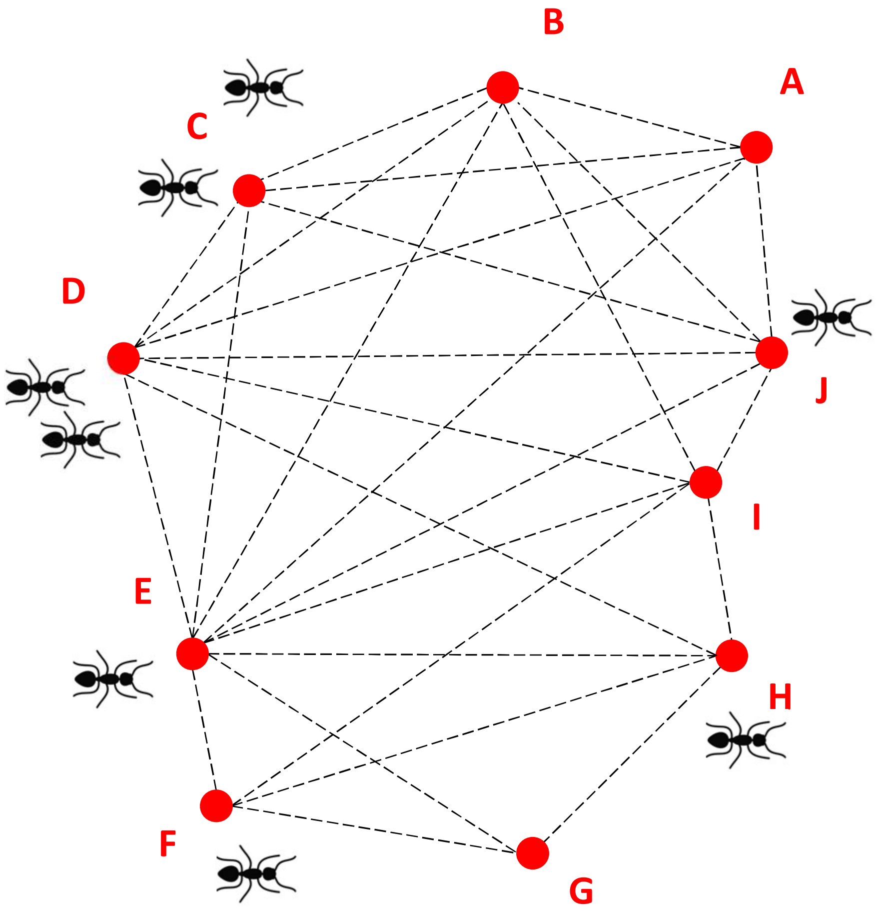
Select Next Node For Each Ant



Completely random moves also possible.

1) **Pick a random destination**
2) **Pick a destination based on heuristic (possibly with some randomness)**

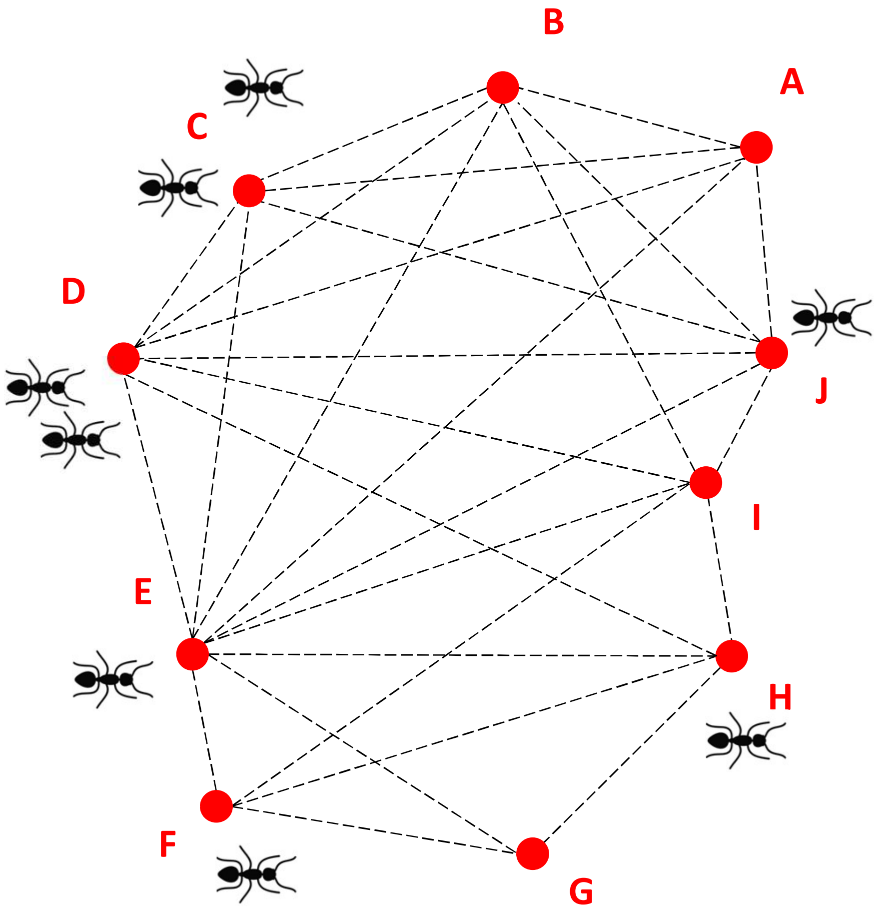
Select Next Node For Each Ant



Each ant will pick its next destination:

- unvisited node
- choice will be based on:
  - pheromone intensities d_{ij} on all available paths
  - heuristic value h_{ij} for all available paths (a distance between nodes)

Select Next Node For Each Ant



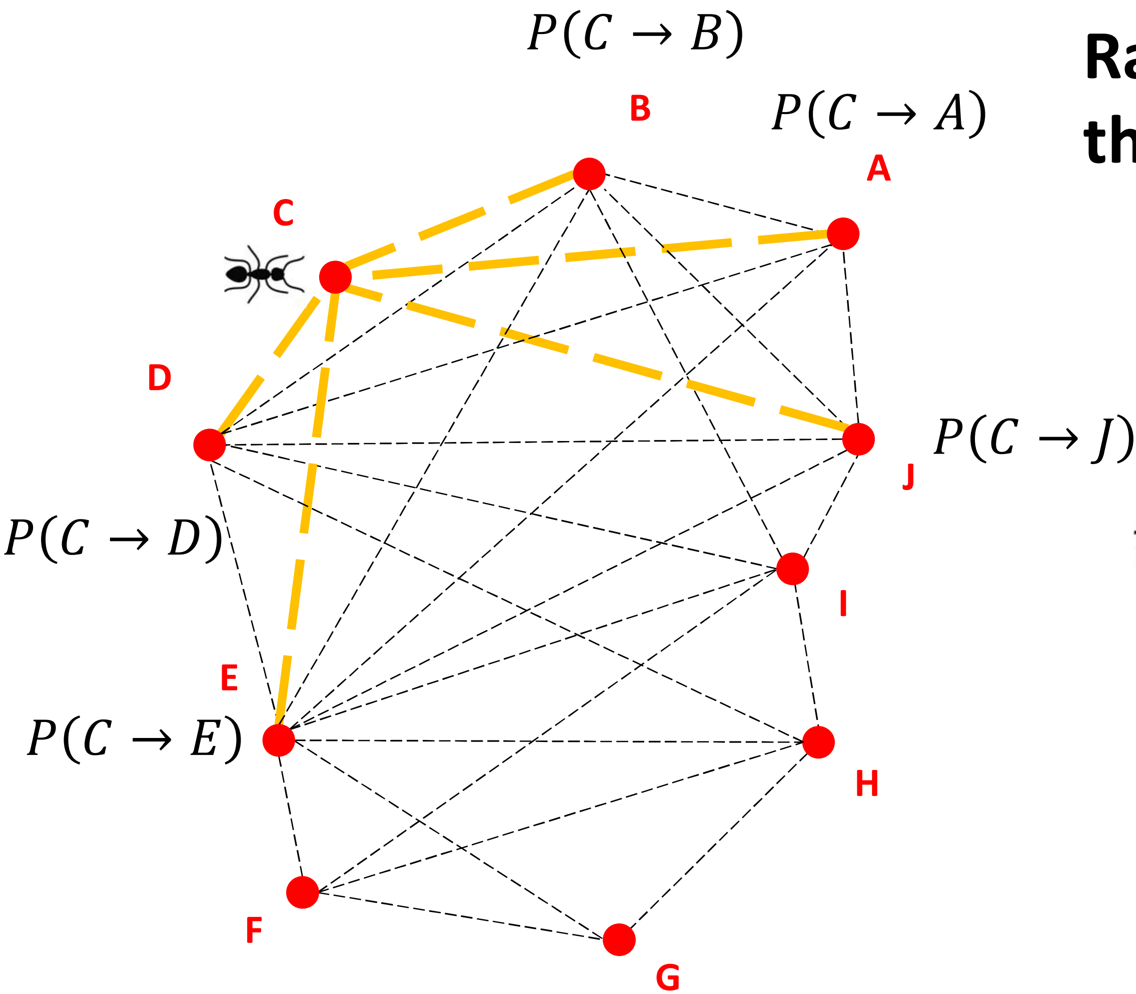
Each ant will pick its next destination:

- unvisited node
- choice will be based on:
  - pheromone intensities d_{ij} on all available paths
  - heuristic value h_{ij} for all available paths (a distance between nodes)

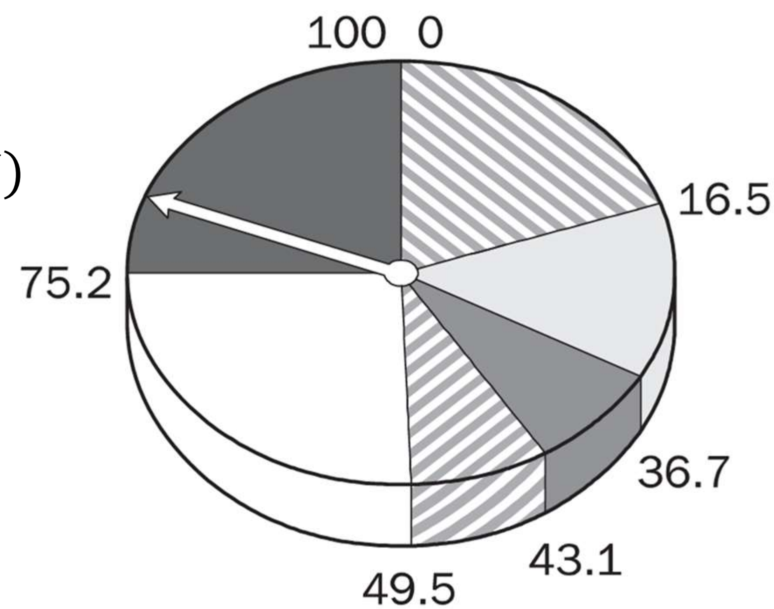
$$P(move\ i \rightarrow j) = \frac{{d_{ij}}^{\alpha} * \frac{1}{h_{ij}}^{\beta}}{\sum_{k=1}^{N\ possible\ destinations} {d_{ik}}^{\alpha} * \frac{1}{h_{ik}}^{\beta}}$$

α – pheromone intensity "weight"
β – heuristic "weight"

Select Next Node For Each Ant



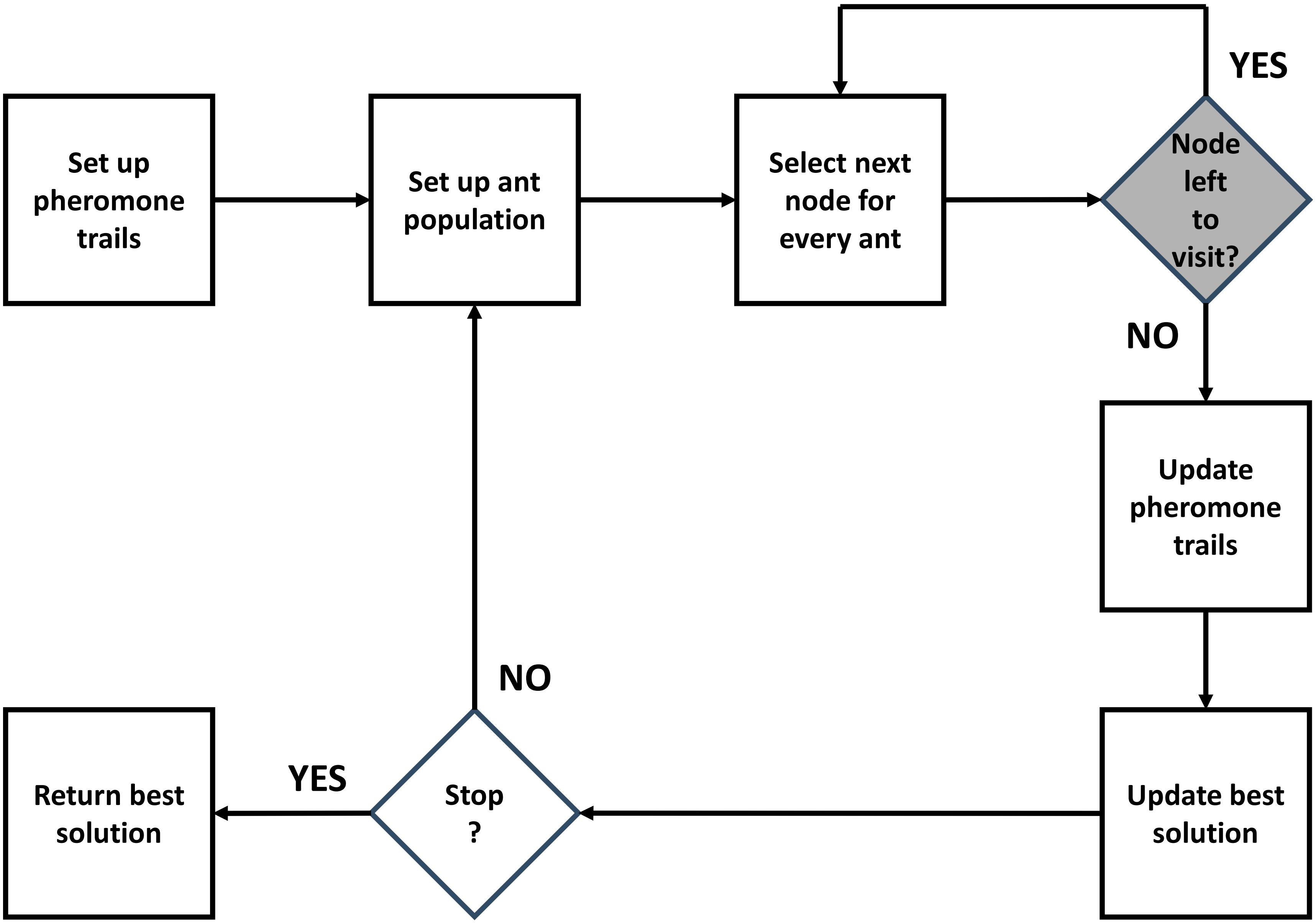
Randomly pick move (using the roulette approach)



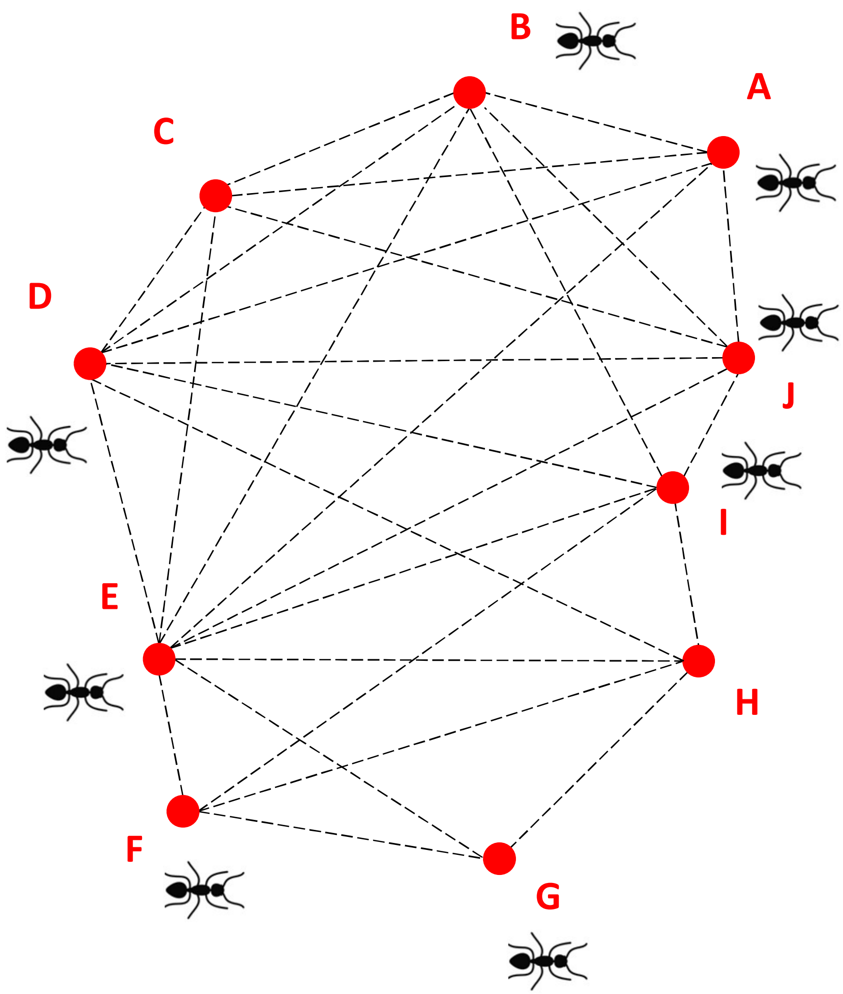
α – pheromone intensity "weight"
β – heuristic "weight"

$$P(move\ i \rightarrow j) = \frac{{d_{ij}}^{\alpha} * \frac{1}{h_{ij}}^{\beta}}{\sum_{k=1}^{N\ possible\ destinations} {d_{ik}}^{\alpha} * \frac{1}{h_{ik}}^{\beta}}$$

TSP with Ant Colony Optimization



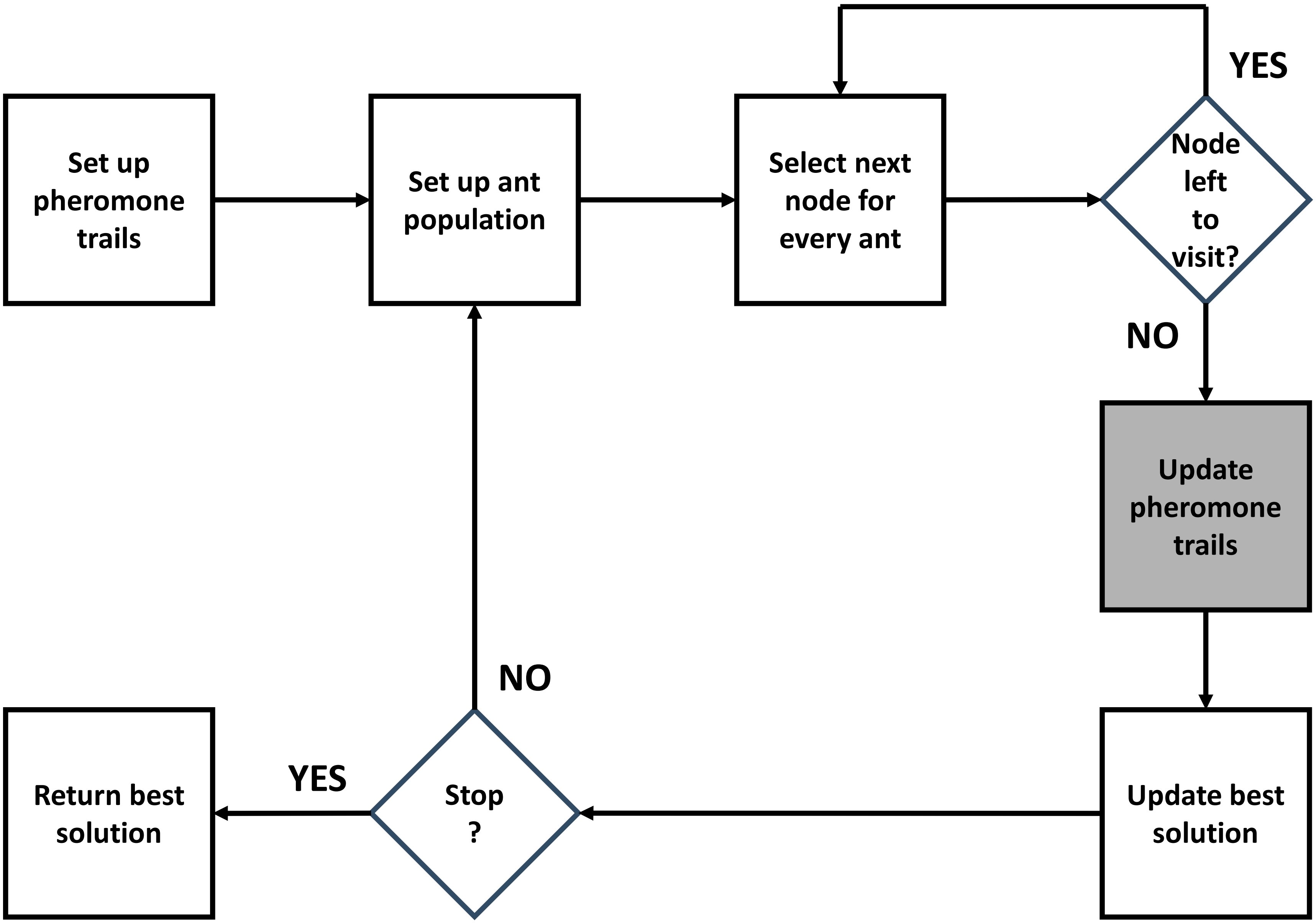
Repeat: Select Next Node For Each Ant



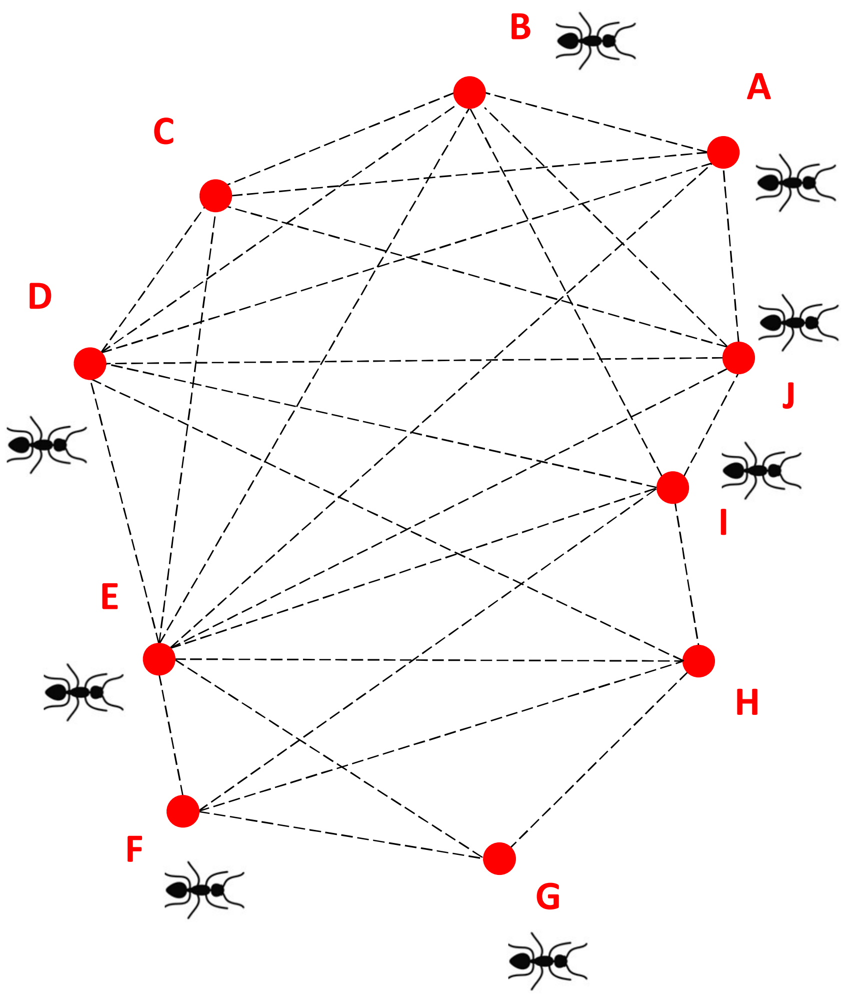
Repeat selecting new nodes until every ant visited every node.

Return to your start location.

TSP with Ant Colony Optimization



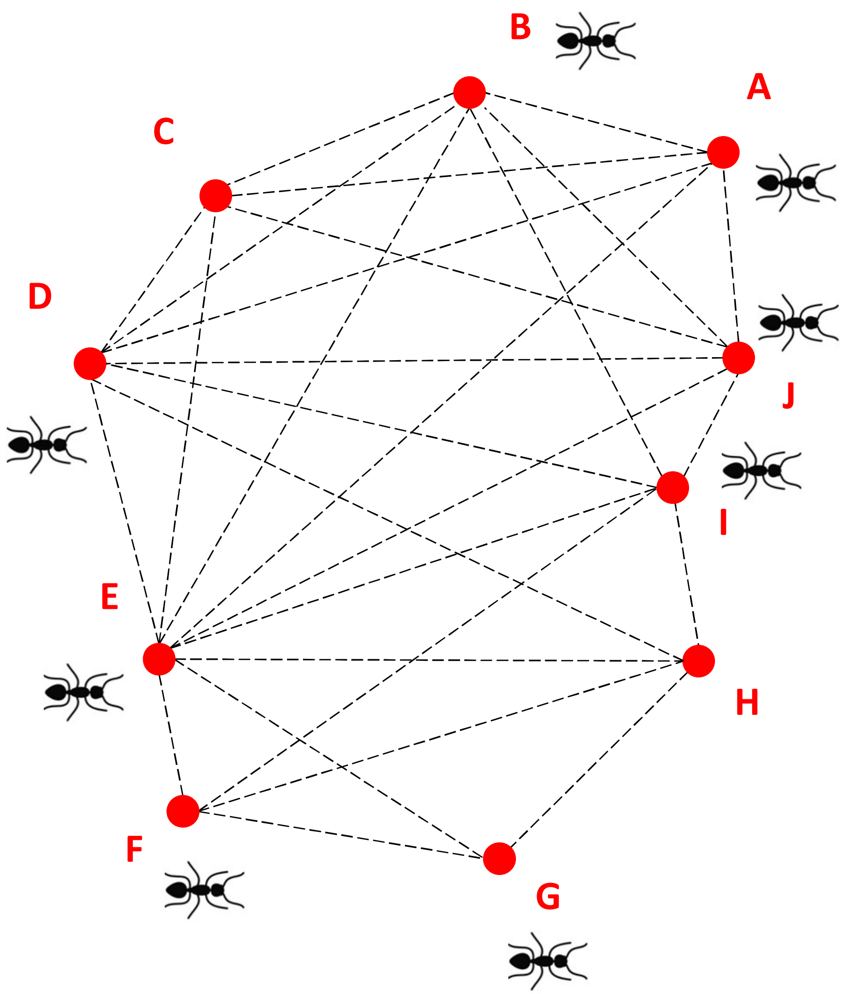
Update Pheromone Trail



Update pheromone trail values for each edge:

- decrease due to pheromone evaporation
- increase if an ant traversed it

Update Pheromone Trail



Update pheromone trail values for each edge:

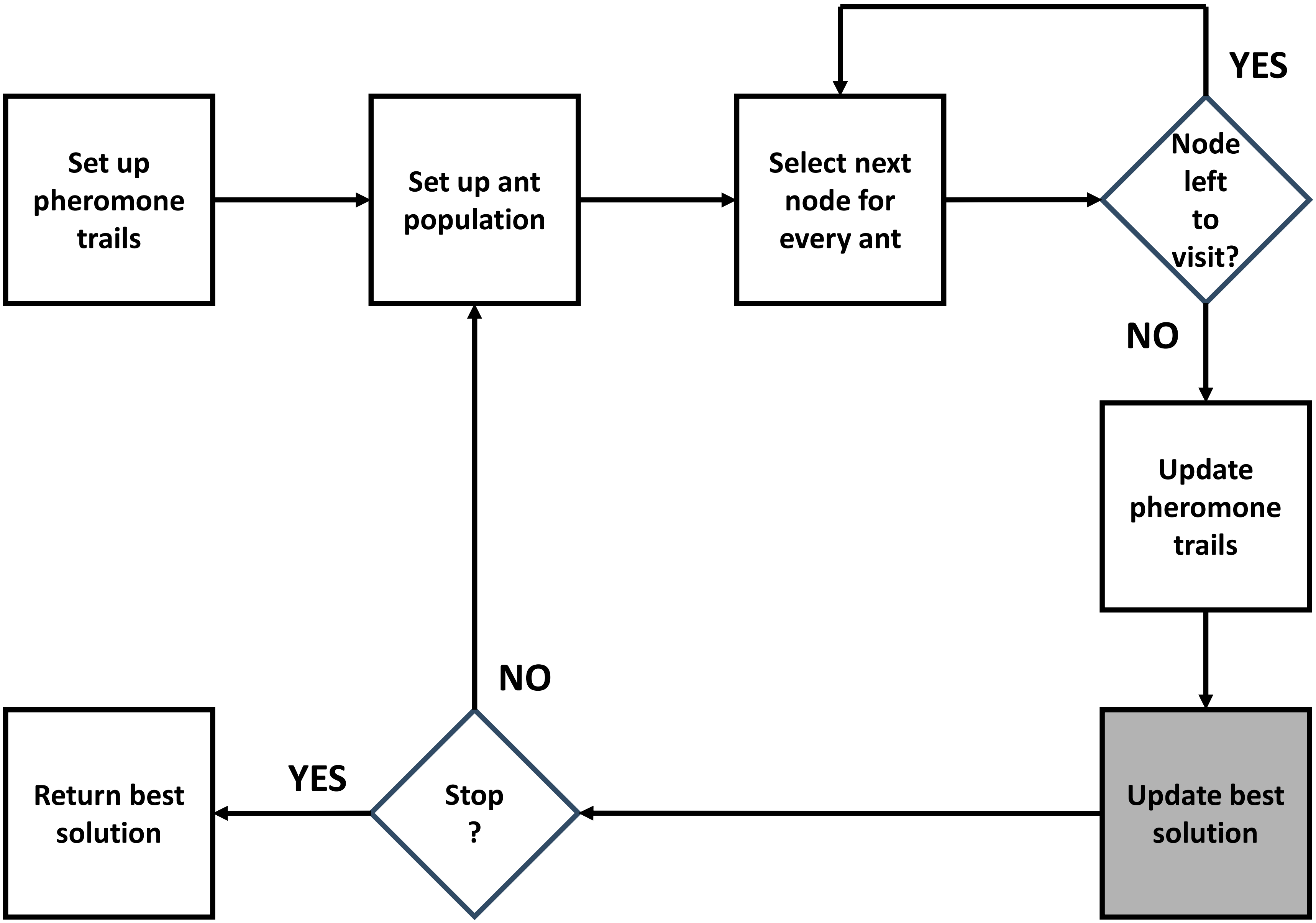
- **decrease due to pheromone evaporation (evaporation rate, for example 0.5)**

$$d_{ij} = d_{ij} * \text{evaporation rate}$$

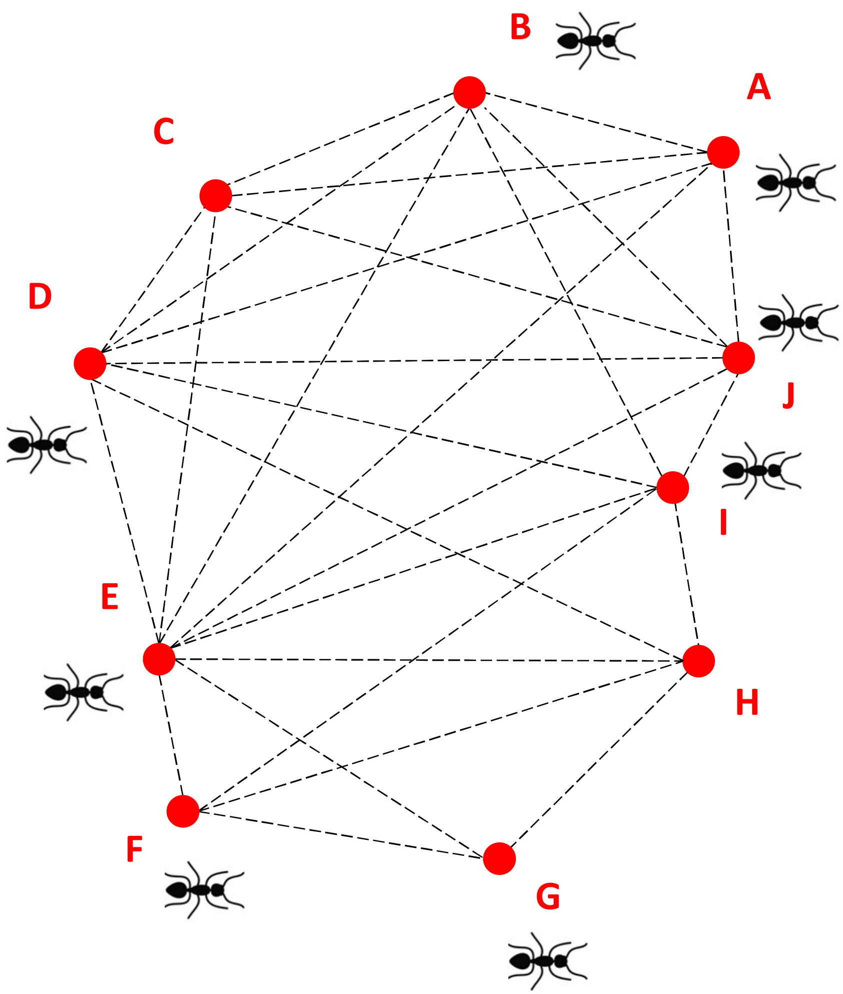
- **increase if an ant traversed it**

$$d_{ij} = d_{ij} + 1/\text{ant fitness}$$

TSP with Ant Colony Optimization

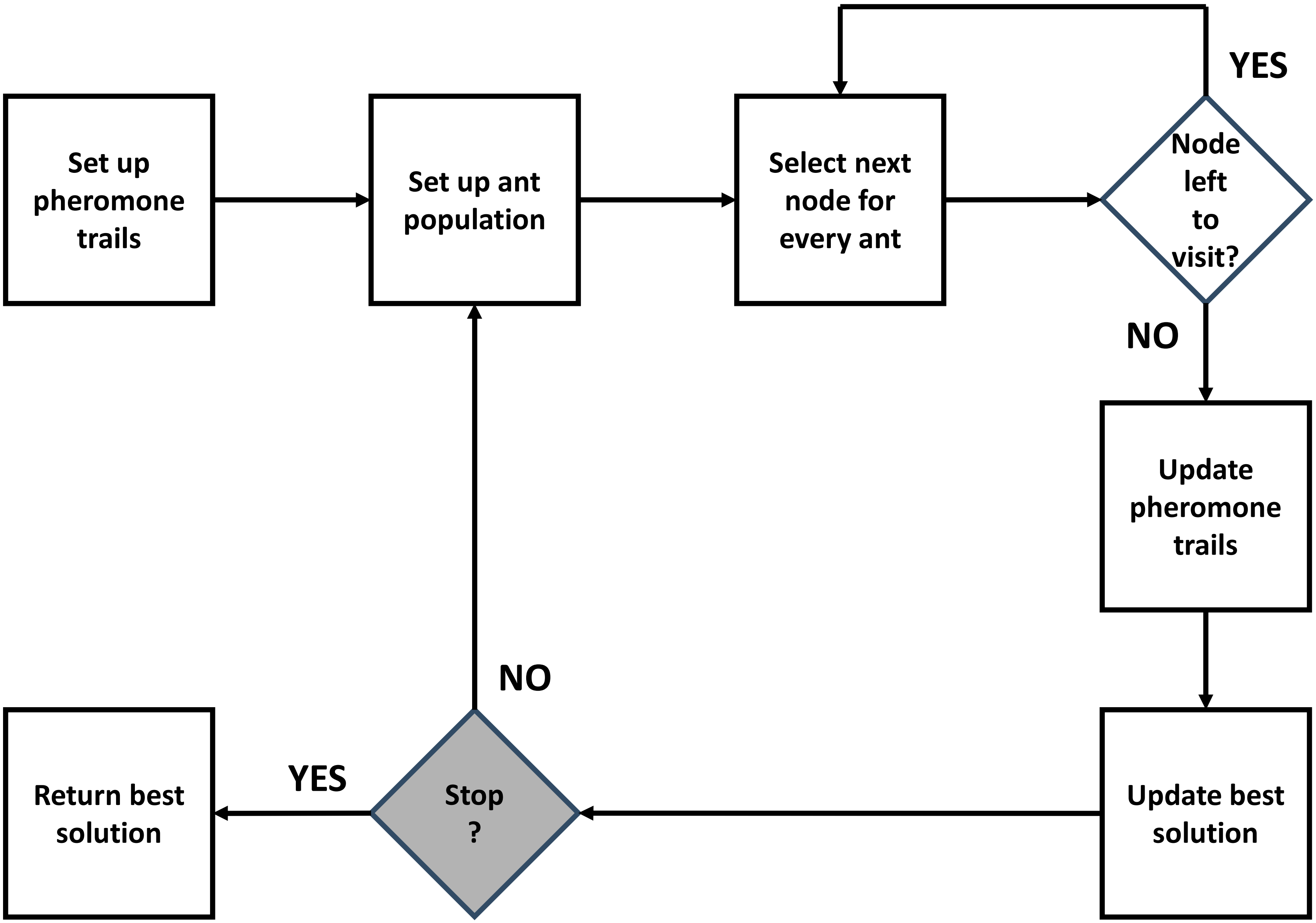


Update Best Solution

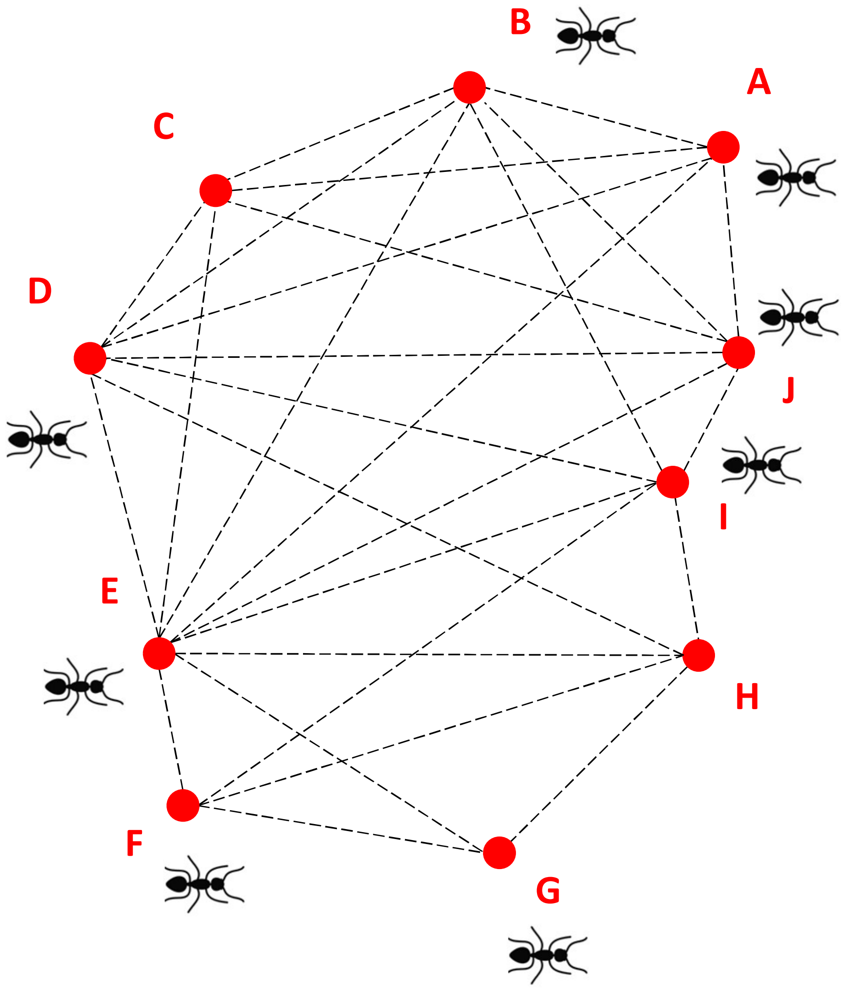


Pick the best solution among all ants

TSP with Ant Colony Optimization



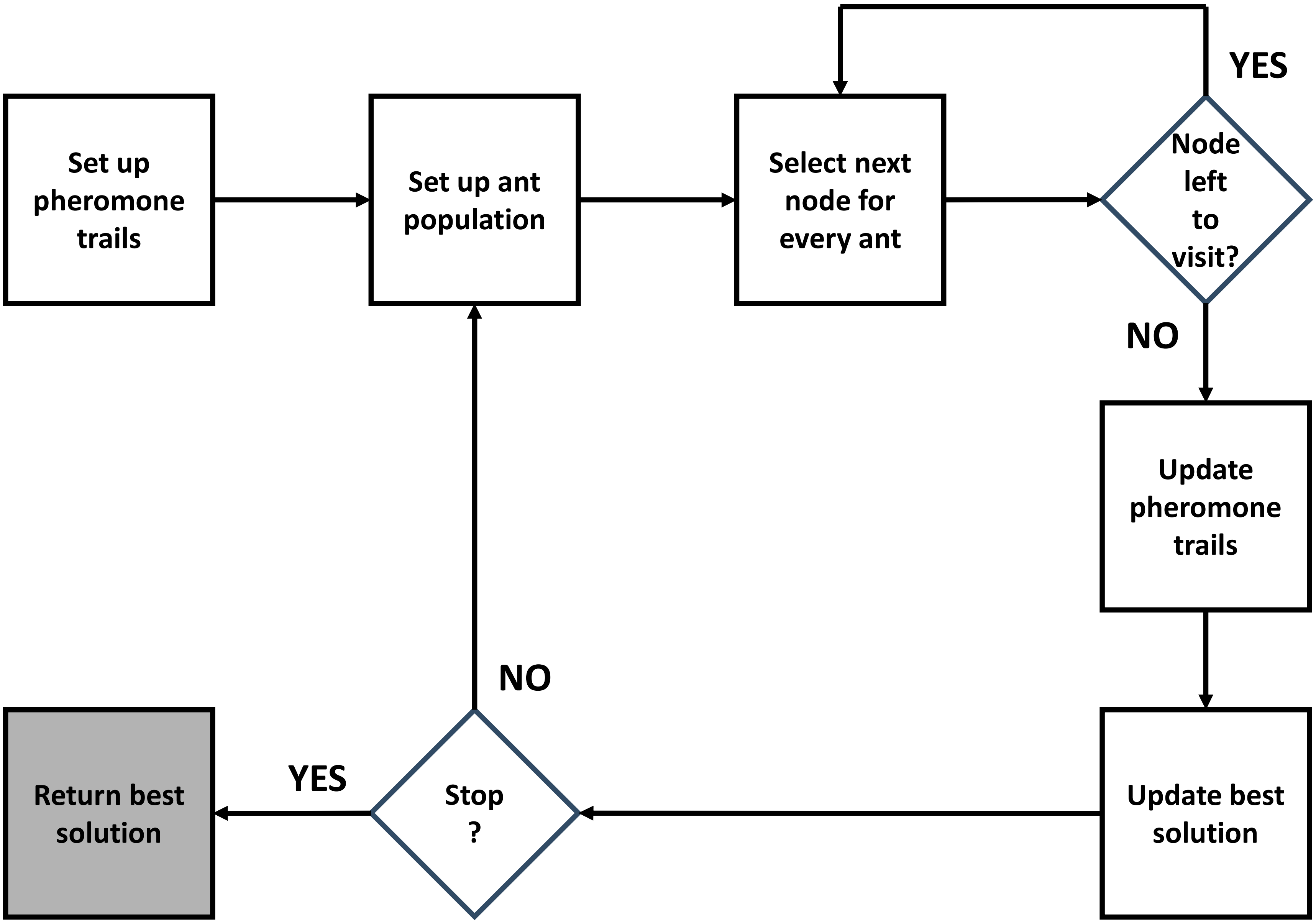
Check The Termination Condition



Check the termination condition:

- number of iterations
- time elapsed
- solution threshold
- solution stagnating for a number of iterations
- etc.

TSP with Ant Colony Optimization



Example: Ant Colony Optimization

https://courses.cs.ut.ee/demos/visual-aco/

ACO Variants / Modifications

- **Ant System (AS)**
- **Elitist Ant System (EAS)**
- **Rank-Based Ant System (ASrank)**
- **Min-Max Ant System (MMAS)**
- **Ant Colony System (ACS)**
- **Approximate Nondeterministic Tree Search (ANTS)**
- **Hyper-Cube Framework for ACO**