MATHEMATICAL PROGRAMMING

CO4

INFINITE DIMENSIONAL OPTIMIZATION

DR.VUDA SREENIVASA RAO

ASSOCIATE PROFESSOR

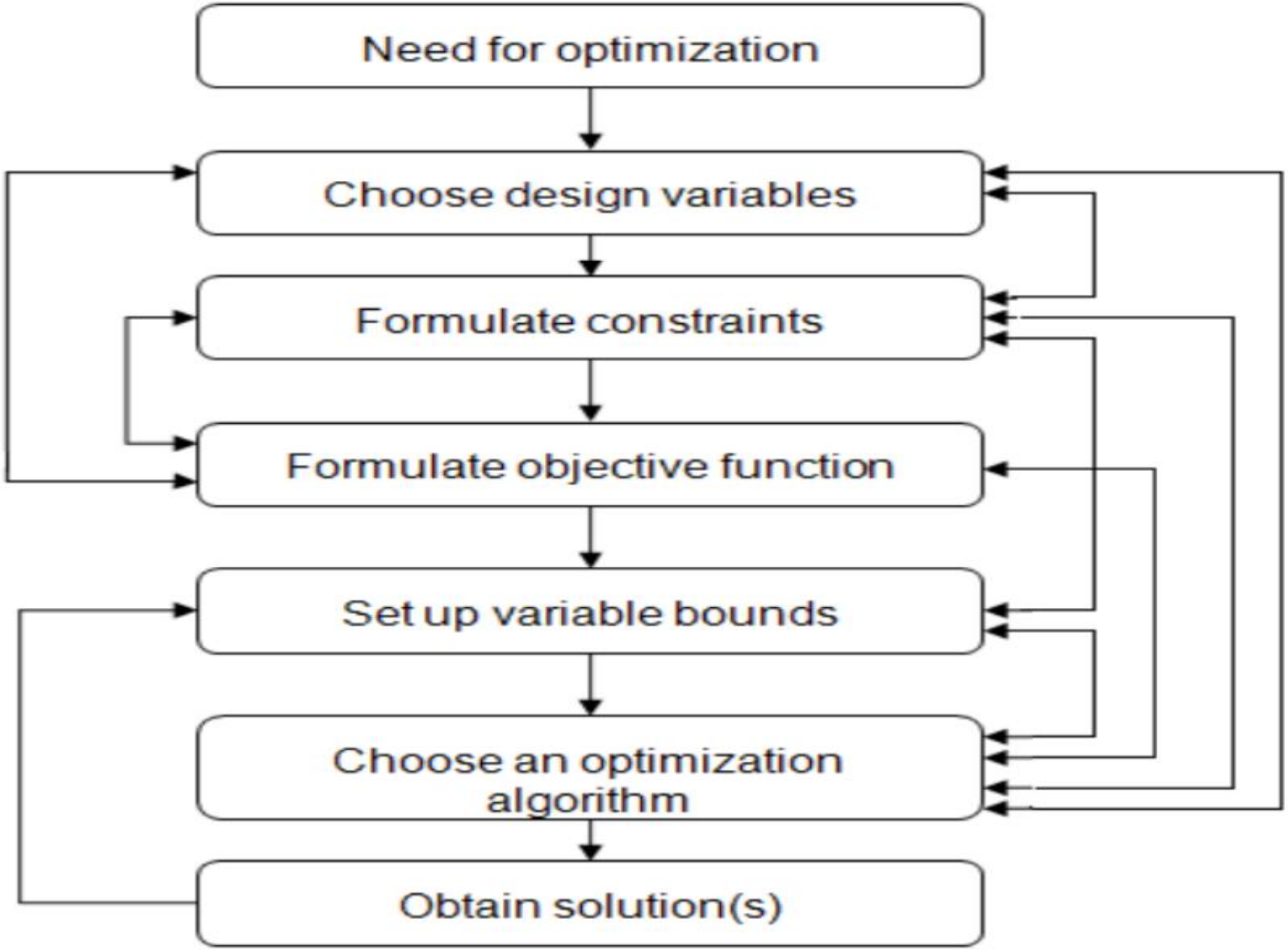
CO4-INFINITE DIMENSIONAL OPTIMIZATION

- Heuristic and Meta heuristics.
- Single solution vs. population-based.
- Parallel meta heuristics.
- Evolutionary algorithms.
- Nature-inspired metaheuristics.
- Genetic Algorithm.
- Ant-colony optimization.
- Particle swarm optimization.
- Simulated annealing.
- Tabu Search.

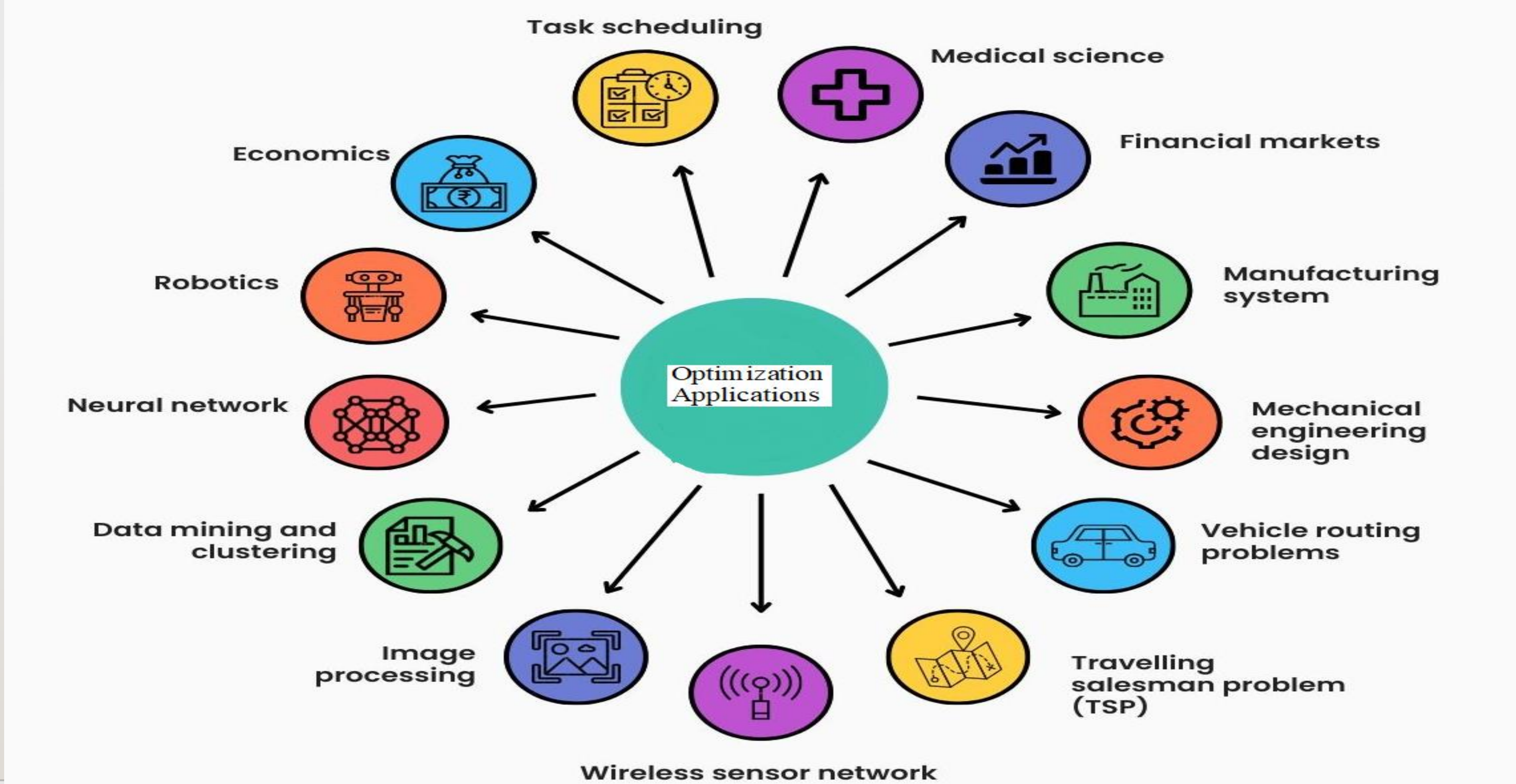
MATHEMATICAL OPTIMIZATION

- Mathematical optimization is the **process of finding the best set of inputs that maximizes (or minimizes) the output of a function.**
- In the field of optimization, the **function being optimized is called the objective function.**

$min\ x_1x_4(x_1+x_2)+x_3$	*Objective Function*
$s.t.\ x_1x_2x_3x_4 \geq 26$	*Inequality Constraint*
$x_1^2+x_2^2+x_3^2+x_4^2=40$	*Equality constraint*
$1 \leq x_1x_2x_3 \geq 25$	*bounds on Variables*
$x_1=(10,12,46)$	*Initial Values*


Need for optimization
Choose design variables
Formulate constraints
Formulate objective function
Set up variable bounds
Choose an optimization algorithm
Obtain solution(s)

APPLICATIONS



SOLUTION STRATEGIES FOR OPTIMIZATION PROBLEMS

Methods to solve Optimization Problems	Nature of Solution
Linear or Non Linear programming	Exact Solution
Branch and Bound	Exact Solution
Heuristic Method	Inexact, Near optimal Solution
Metaheuristic Method	Inexact, Near optimal Solution

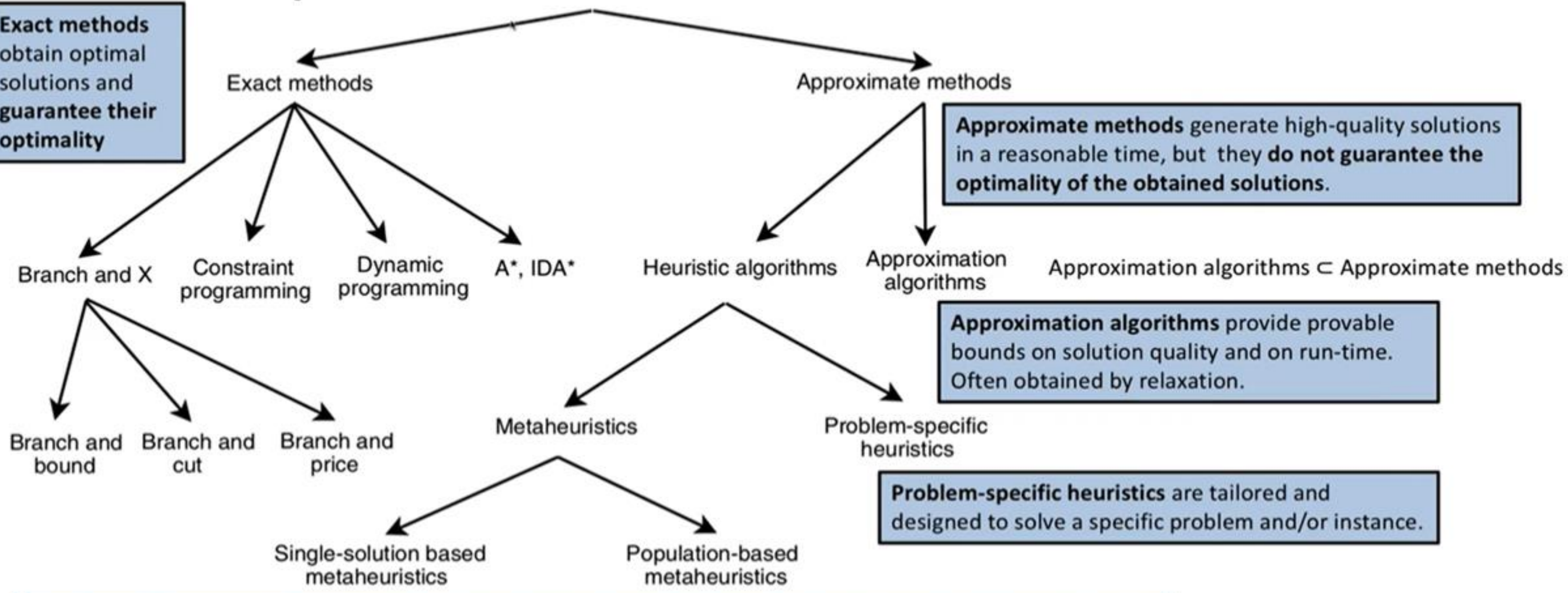
- In Heuristic and Metaheuristic method, we make **a trade-off between solution quality and computational time.**

HEURISTIC METHOD VS METAHEURISTIC METHOD

	Heuristic Method	Metaheuristic Method
Nature	Deterministic	Randomization + Heuristic
Type	Algorithmic	Nature Inspired, Iterative
Example	Nearest Neighbourhood Travelling salesman problems	Genetic Algorithm for Travelling Salesman Problems
Nature of Solution	Inexact, Near optimal Solution	Inexact, Near optimal Solution

Metaheuristics

Optimization methods



Exact methods obtain optimal solutions and **guarantee their optimality**

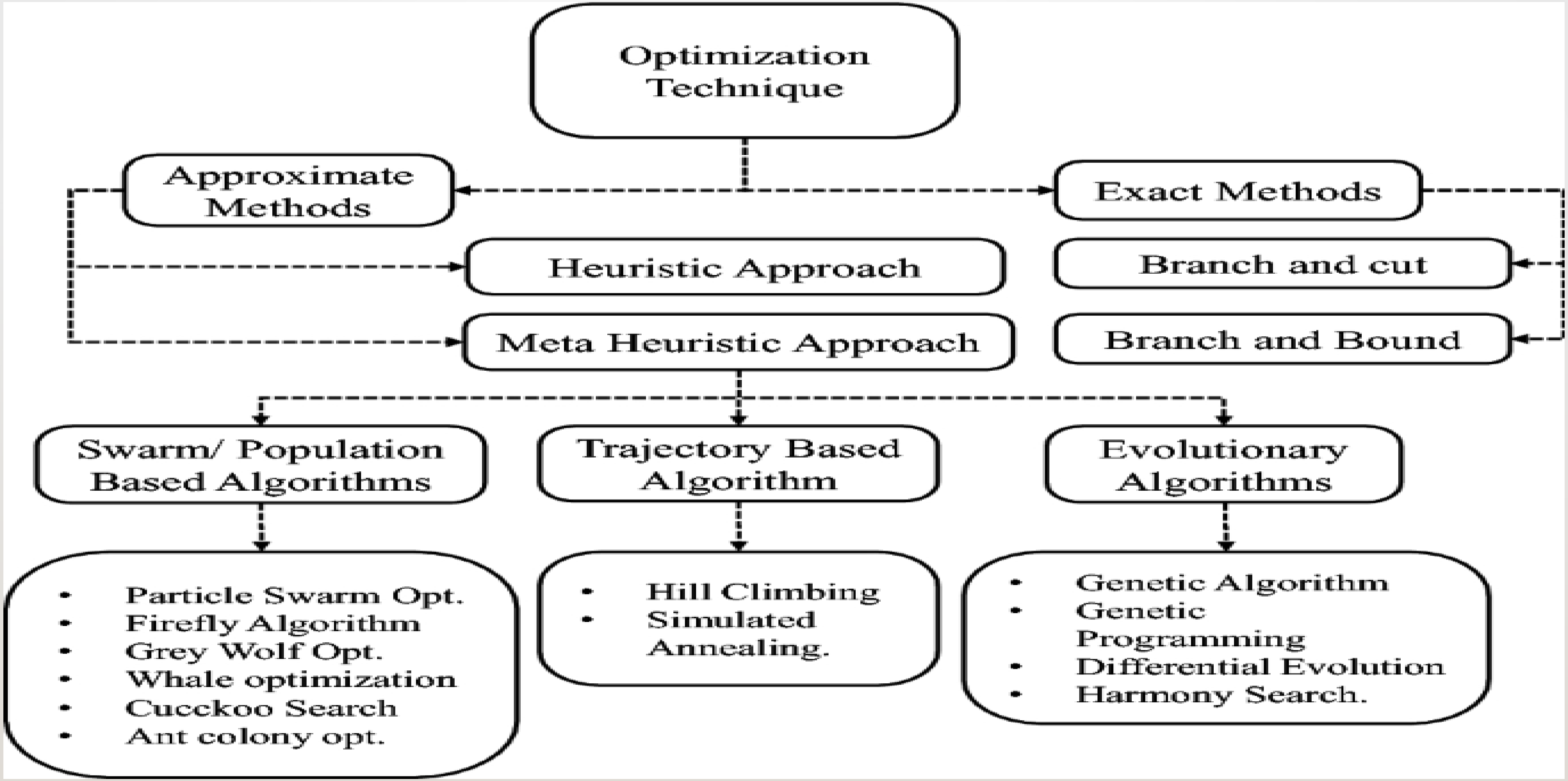
Approximate methods generate high-quality solutions in a reasonable time, but they **do not guarantee the optimality of the obtained solutions.**

Approximation algorithms provide provable bounds on solution quality and on run-time. Often obtained by relaxation.

Problem-specific heuristics are tailored and designed to solve a specific problem and/or instance.

Metaheuristics are **general-purpose algorithms** that can be applied to solve almost any optimization problem. Unlike approximation algorithms, metaheuristics **do not provide any bound** on how close the obtained solutions is to the optimal one. Unlike exact methods, metaheuristics **allow to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time**

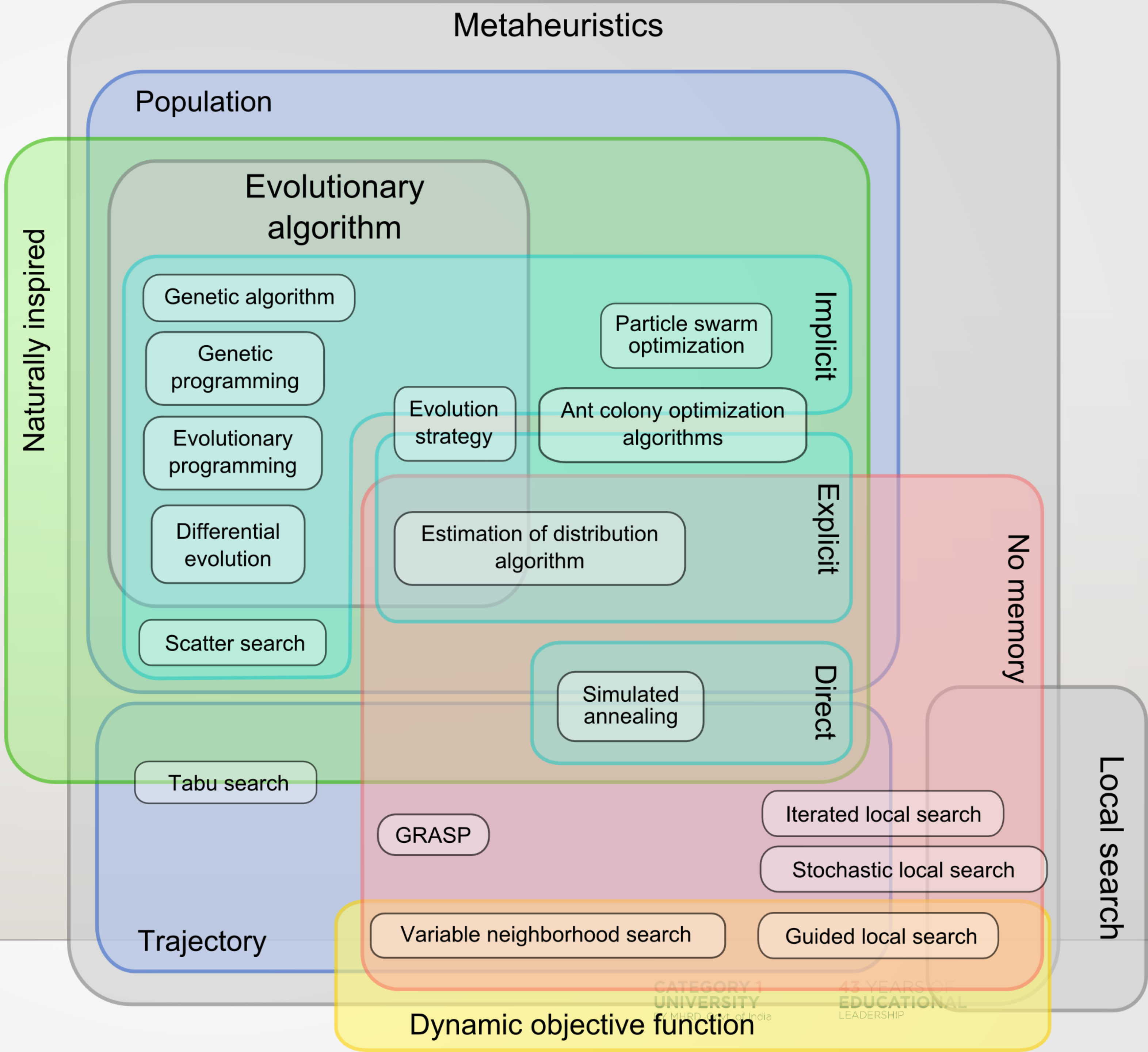
Source: Talbi, E. G. (2009). Metaheuristics: from design to implementation (Vol. 74). John Wiley & Sons.


Optimization Technique
Approximate Methods
Exact Methods
Heuristic Approach
Branch and cut
Meta Heuristic Approach
Branch and Bound
Swarm/ Population Based Algorithms
Trajectory Based Algorithm
Evolutionary Algorithms
• Particle Swarm Opt.
• Firefly Algorithm
• Grey Wolf Opt.
• Whale optimization
• Cucckoo Search
• Ant colony opt.
• Hill Climbing
• Simulated Annealing.
• Genetic Algorithm
• Genetic Programming
• Differential Evolution
• Harmony Search.

Taxonomies

Many classification criteria may be used for metaheuristics. The most common are:

- Nature inspired vs non-nature inspired
- Memory usage vs memoryless methods
- Deterministic vs stochastic
- Iterative vs greedy
- Population-based search vs single-solution based search



METAHEURISTIC

- The word **‘meta’** means **higher level**, where as the word **‘heuristics**’ means **to find**.
  - **In computer science**, metaheuristic designates a computational method that optimizes problem by iteratively trying to improve a candidate solution with regard to given measure of quality.
- Metaheuristic optimization is the best approach to optimizing such **non-convex functions.**
- Metaheuristics **do not guarantee an optimal solution**.
- Metaheuristics implement **some form of stochastic optimization.**

METAHEURISTICS HAVE FUNDAMENTAL CHARACTERISTICS :

- Heuristics can be employed by a metaheuristic as a domain-specific knowledge which is dominated by the upper-level strategy.
- Metaheuristics are not for a particular problem.
- Metaheuristics are usually approximate.
- Metaheuristics essentially can be described by abstraction level.
- Metaheuristics usually allow an easy parallel implementation.
- Metaheuristics extend from basic local search to advanced learning techniques.
- Metaheuristics may incorporate various mechanisms in order to avoid premature convergence.

(DEEMED TO BE UNIVERSITY)

Algorithmic framework for metaheuristics

```
Create one or more initial solutions
  While (stopping criterion not satisfied) do
      If exploit then
          Create new solution by exploitation step;
      Else
          Create new solution by exploration step;
      End
      Update best found solution ;
  End
  Return best found solution;
```

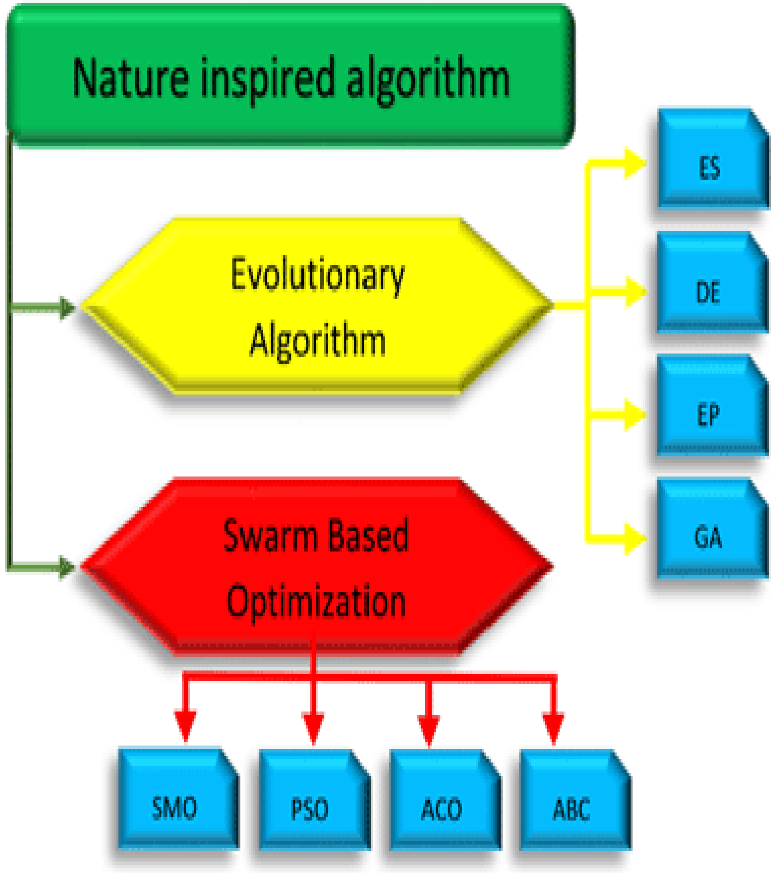
CLASSIFICATION OF METAHEURISTICS

- ***Nature-inspired vs. non-nature inspired***
- ***Population-based vs. single point search***
- ***Dynamic vs. static objective function***
- ***Memory usage vs. memory-less methods***

Nature Inspired Algorithms

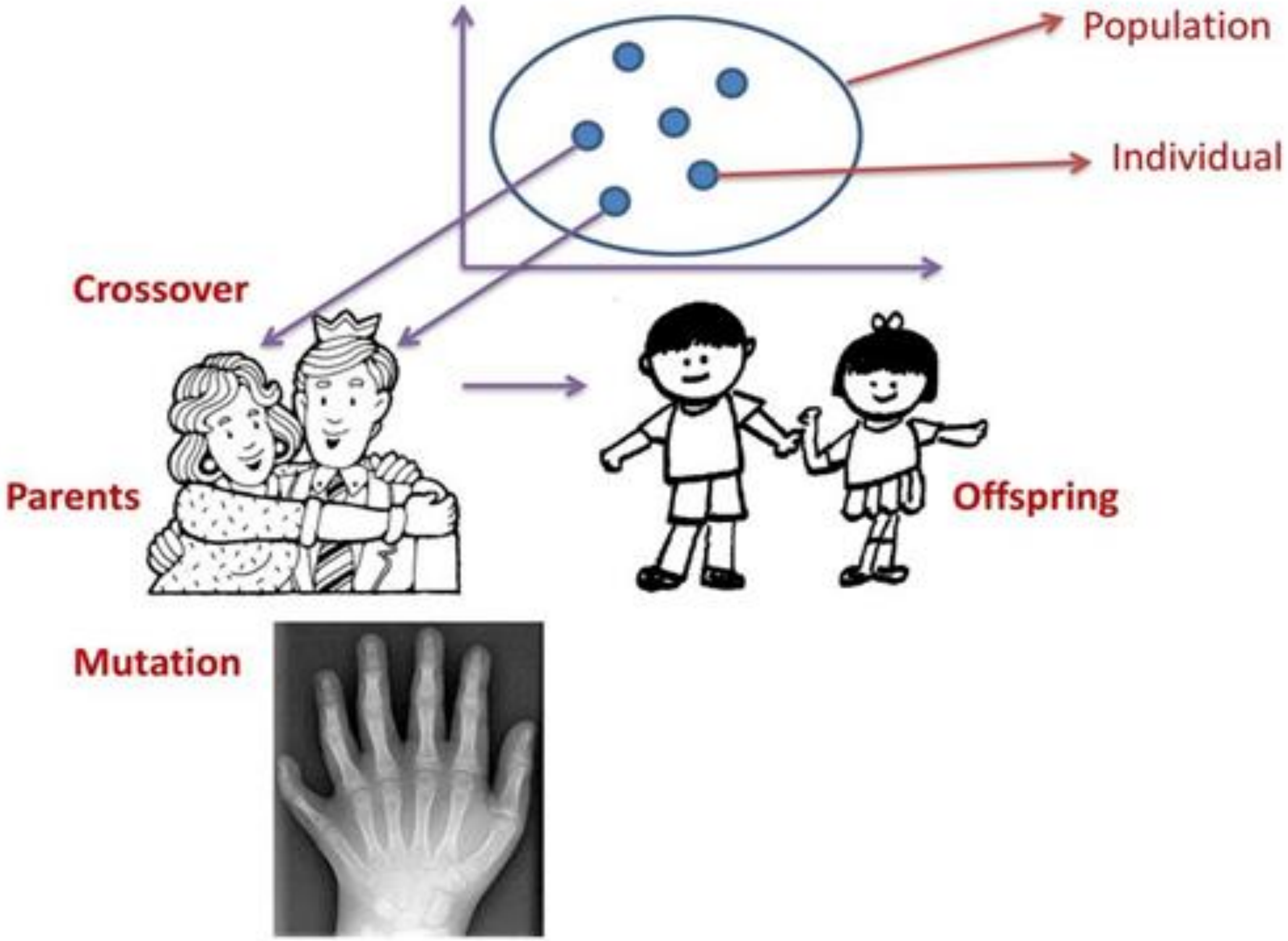
- Nature provide some of the efficient ways to solve problems
  - Algorithms imitating processes in nature/inspired from nature – **Nature Inspired Algorithms**.
- What type of **problems**?
  - Aircraft wing design

Nature Inspired Algorithms for Optimization



GA→ Genetic Algorithm
ES → Evolution Strategy
DE → Differential Evolution
EP → Evolution Programming
ABC → Artificial Bee Colony
ACO→ Ant Colony Optimization
PSO → Particle Swarm Optimization
SMO→ Sequential Minimal Optimization

Evolutionary Algorithms



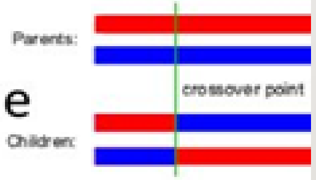
Evolutionary Algorithms

- **Terminologies**
  1. **Individual** - carrier of the genetic information (chromosome). It is characterized by its state in the search space, its fitness (objective function value).
  2. **Population** - pool of individuals which allows the application of genetic operators.
  3. **Fitness function** - The term "fitness function" is often used as a synonym for objective function.
  4. **Generation** - (natural) time unit of the EA, an iteration step of an evolutionary algorithm.

FEATURES OF THE EAS

Evolutionary Algorithms

- Selection - Roulette wheel, Tournement, steady state, etc.
  - Motivation is to preserve the best (make multiple copies) and eliminate the worst
- Crossover – simulated binary crossover, Linear crossover, blend crossover, etc.
  - Create new solutions by considering more than one individual
  - Global search for new and hopefully better solutions



- Mutation – Polynomial mutation, random mutation, etc.
  - Keep diversity in the population
  - 010110 →010100 (bit wise mutation)

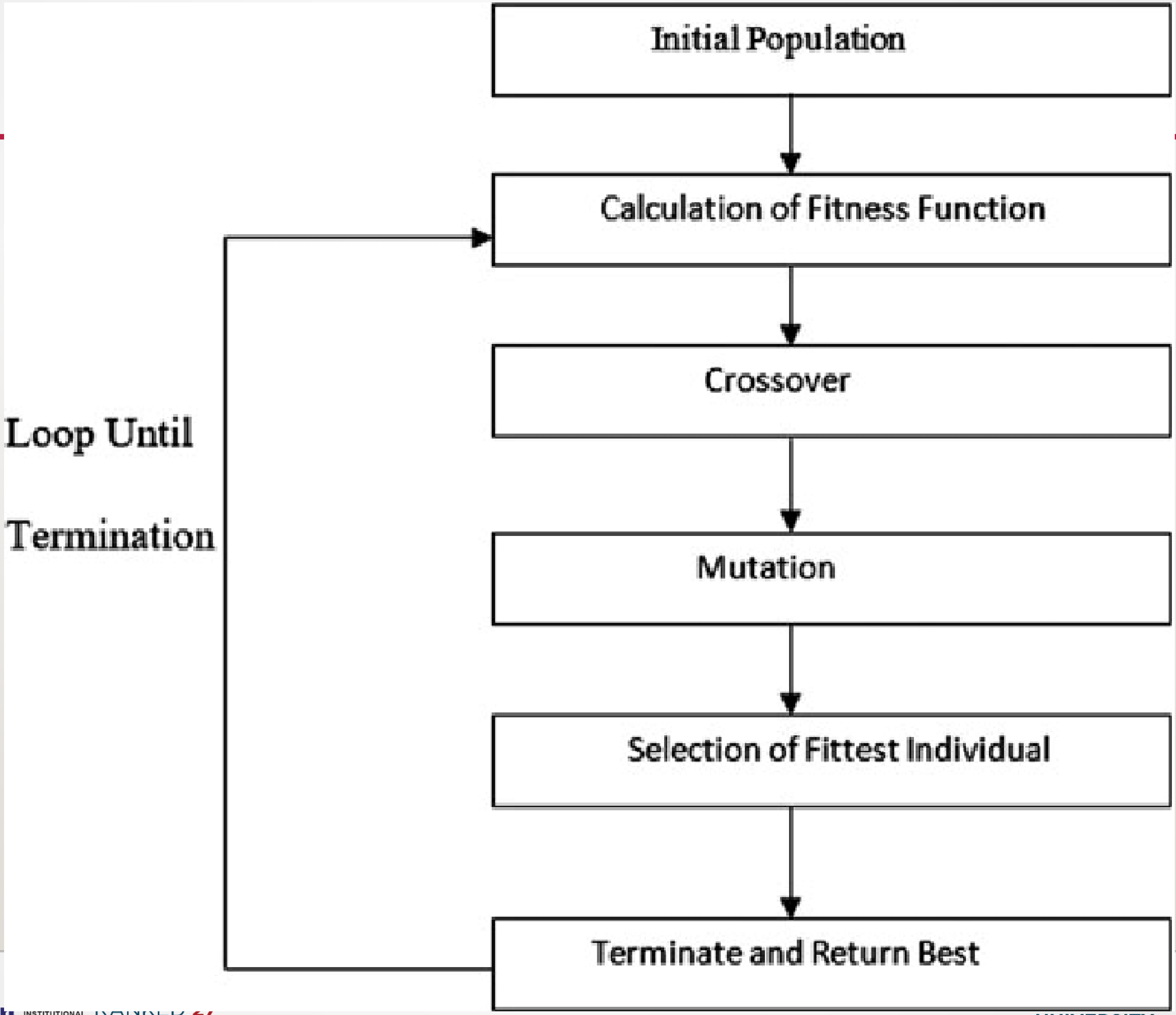
Evolutionary Algorithms

- Concept of **exploration vs exploitation.**
- Exploration – Search for promising solutions
  - Crossover and mutation operators
- Exploitation – preferring the good solutions
  - Selection operator
- **Excessive** exploration – Random search.
- **Excessive** exploitation – Premature convergence.

WHAT IS GA

- A **genetic algorithm** (or **GA**) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- Genetic algorithms are categorized as global search heuristics.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Basic Structure of GA



EXAMPLE

- **Using Genetic algorithm maximize the function**

$f(x) = x^2$

with x in interval [0, 31] i.e., x = 0,1,2,…..30,31.

- Select Encoding Technique: **Binary encoding Technique**
- The minimum value is **0** and maximum value is **31**
- To represent the values, use 5-digit binary code numbers between 0 to 31

0 (00000) to 31 (11111) is obtained

- The objective function is to be maximized (**$f(x) = x^2$**)

String No.	Initial Population (Randomly selected)	X value	Fitness value $f(x) = x^2$	Prob. $f(x)/\Sigma f(x)$	%prob.	Expected count $f(x)/Avg(\Sigma f(x))$	Actual count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	361	0.3126	31.26	1.2502	1
Sum			1155	1.0	100	4	4
Average			288.75	0.25	25	1	1
Max.			**625**	0.5411	51.11	2.1645	2

String No.	Mating pool	Crossover point	Offspring after crossover	X value	Fitness value $f(x) = x^2$
1	01100	4	01101	13	169
2	11001	4	11000	24	576
3	11001	2	11011	27	729
4	10011	2	10001	17	289
Sum					1763
Average					440.75
Max.					**729**

String No.	Offspring after crossover	Mutation chromosome for flipping	Offspring after mutation	X value	Fitness value f(x) = x^2
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00101	10100	20	400
Sum					2546
Average					636.5
Max.					**841**

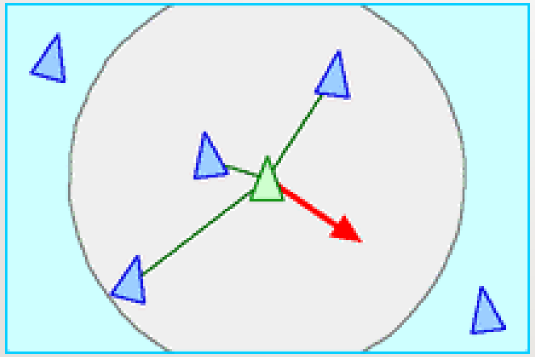
PARTICLE SWARM OPTIMIZATION (PSO)

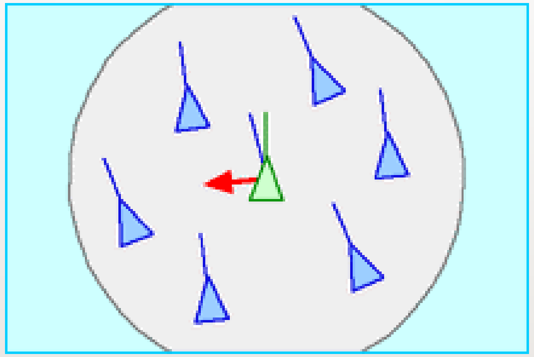
INTRODUCTION TO THE PSO: <u>**ORIGINS**</u>

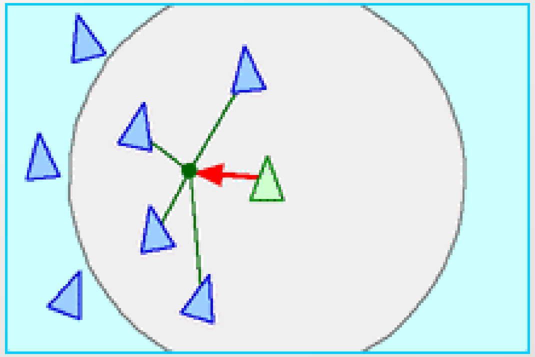
- <u>**Inspired from the nature**</u> **social behavior and dynamic movements with communications of insects, birds and fish.**

CONT..

- **In 1986, Craig Reynolds described this process in 3 simple behaviors:**







Separation

avoid crowding local flockmates

Alignment

move towards the average heading of local flockmates

Cohesion

move toward the average position of local flockmates

CONT….

- **Application to optimization: <u>Particle Swarm Optimization</u>**
- **Proposed by James Kennedy & Russell Eberhart (1995)**
- **Combines <u>self-experiences</u> with <u>social experiences</u>**

INTRODUCTION TO THE PSO: CONCEPT

- Uses a number of agents (particles) that constitute a swarm moving around in the search space looking for the best solution.
- Each particle in search space adjusts its "flying" according to its own flying experience as well as the flying experience of other particles.

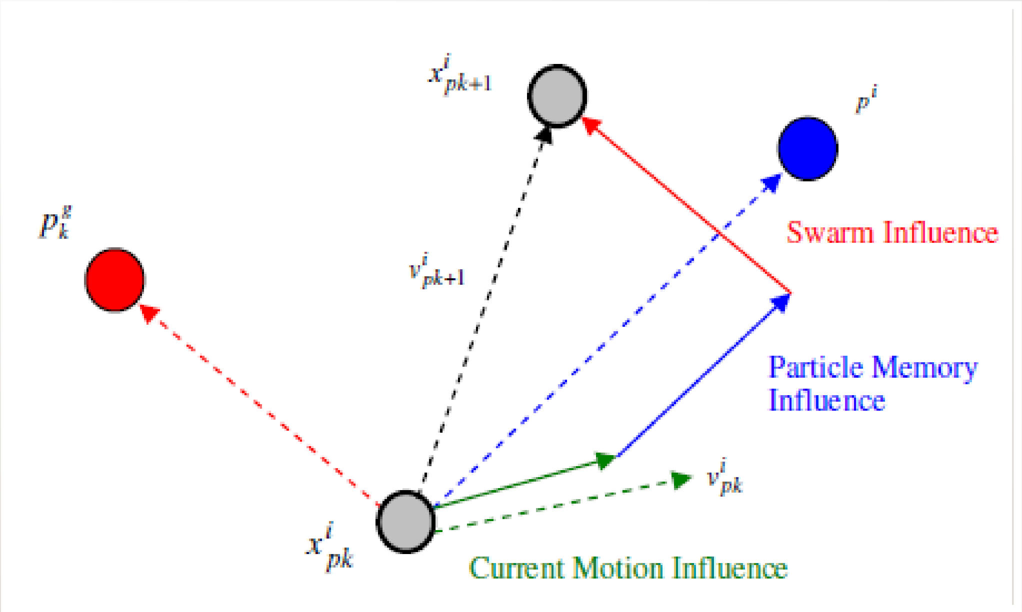
CONT....

- **Collection of flying particles (swarm) - Changing solutions**
- **Search area - Possible solutions**
- **Movement towards a promising area to get the global optimum.**
- **Each particle keeps track:**
  - **its best solution, personal best, *pbest***
  - **the best value of any particle, global best, *gbest***

CONTD...

- **Each particle adjusts its travelling speed dynamically corresponding to the flying experiences of itself and its colleagues.**

- **Each particle modifies its position according to:**
  - **its current position**
  - **its current velocity**
  - **the distance between its current position and p_{best}**
  - **the distance between its current position and g_{best}**



PARTICLE SWARM optimization (PSO) ALGORITHM

- Basic Algorithm of PSO

  1. **Initialize the swarm from the solution space.**
  2. **Evaluate fitness of each particle.**
  3. **Update individual and global bests.**
  4. **Update velocity and position of each particle.**
  5. **Go to step 2, and repeat until termination condition.**

UPDATE VELOCITY AND POSITION OF EACH PARTICLE.

- **Velocity of particle**

$$v(t+1) = \{V(t) + c_1 * r_1 * (P_{best} - x) + c_2 * r_2 * (G_{best} - x)\}$$

Where

x : particle's position, *v*: path direction

r_1, r_2 are the random numbers in the range of (0, 1)

c_1: weight of local information, c_2: weight of global information

p_{best}: best position of the particle

g_{best}: best position of the swarm

- **Position of particle:** $x(t+1) = x(t) + v(t+1)$

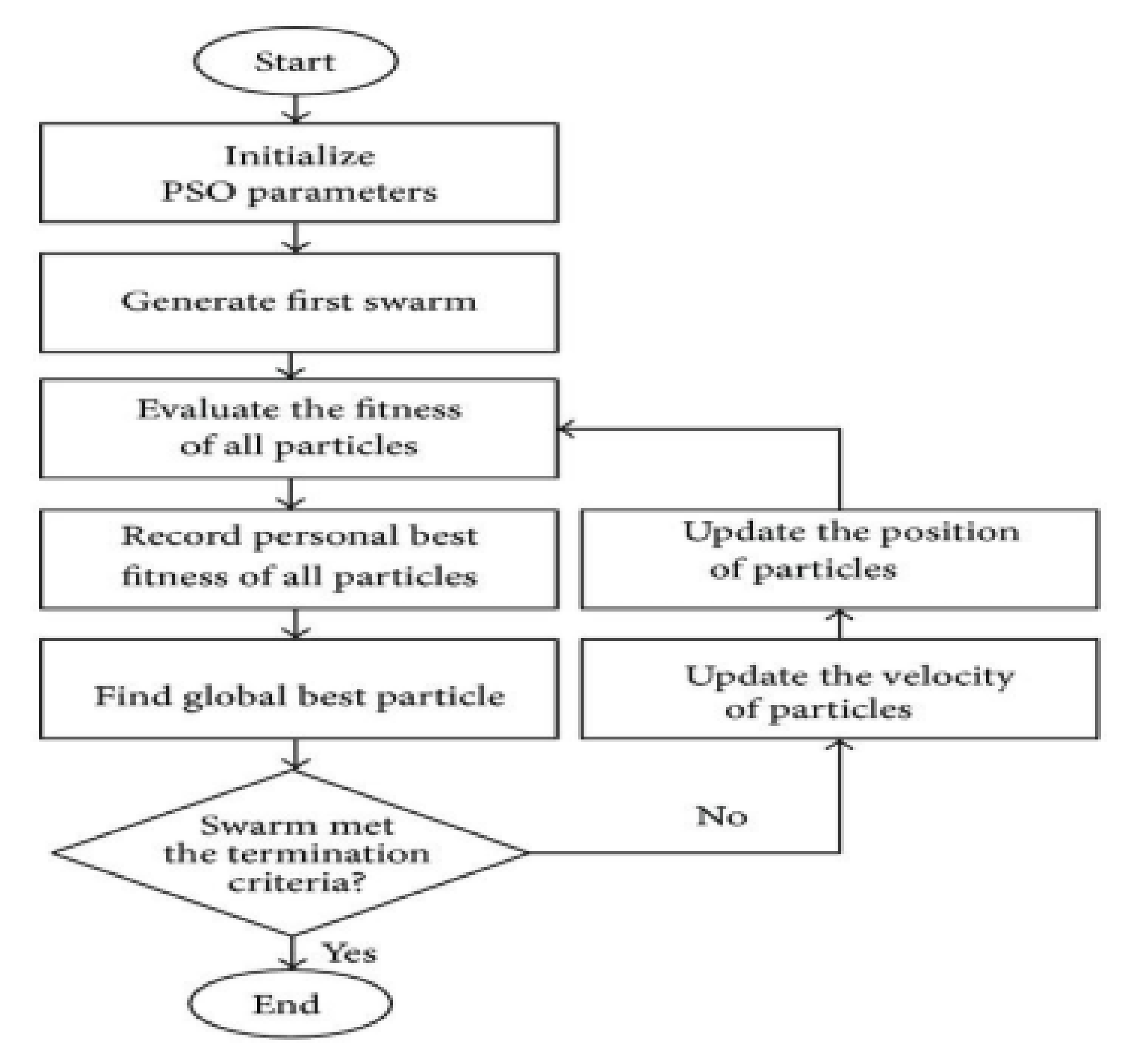
PSO ALGORITHM - PARAMETERS

- Number of particles usually between 10 and 50
- C_1 is the importance of personal best value
- C_2 is the importance of neighborhood best value
- Usually $C_1 + C_2 = 4$ (empirically chosen value)
- If velocity is too low → algorithm too slow
- If velocity is too high → algorithm too unstable

PROBLEM ANALYSIS

1. Size of a swarm.
2. How to generate initial particles with position and velocity.
3. Finding fitness function.
4. Finding P_{best} and G_{best}.
5. Updating Velocity. (values of C_1,C_2,W, etc.)
6. limits for velocity (Vmax,Vmin)
7. Updating position.
8. Terminating condition.

Flow chart of Algorithm



DEFINE THE PROBLEM

Find the maximum of the function

$$f(x) = -x^2 + 5x + 20$$

- **with $-10 \le x \le 10$ using the PSO algorithm.**

PROBLEM ANALYSIS

- 1 Size of a swarm.
- 2 How to generate initial particles with position and velocity.
- 3 Finding fitness function.
- 4 Finding P_{best} and G_{best}.
- 5 Updating Velocity. (values of C_1,C_2,W, etc.)
- 6 limits for velocity (Vmax,Vmin)
- 7 Updating position.
- 8 Terminating condition.
- 9 ....

INITIALIZATION

- Use 9 particles with the initial positions x_1= -9.6, x_2= -6, x_3= -2.6, x_4= -1.1, x_5 = 0.6, x_6 = 2.3, x_7 =2.8, x_8 = 8.3 and x_9 =10.

Particle Number	Particles Initial Position	Evaluate the objective function $f(x) = -x^2 + 5x + 20$
1	-9.6	-120.16
2	-6	-46
3	-2.6	0.24
4	-1.1	13.29
5	0.6	22.64
6	2.3	26.21
7	2.8	26.16
8	8.3	-7.39
9	10	-30

PARTICLE VELOCITY INITIALIZATION

- Let $C_1 = C_2 = 1$ and set initial velocities of the particles to zero.

Particles	v_1^0	v_2^0	v_3^0	v_4^0	v_5^0	v_6^0	v_7^0	v_8^0	v_9^0
Velocities	0	0	0	0	0	0	0	0	0

- Step2: Set the iteration no as t=0+1 and go to step 3
- Step 3. Find the personal best (P_{best}) for every particle.

$$P_{best,\ i}^{t+1} = \begin{cases} P_{best,\ i}^{t}\ iff_i^{t+1} > P_{best,\ i}^{t} \\ x_i^{t+1}\ iff_i^{t+1} \le P_{best,\ i}^{t} \end{cases}$$

$$P_{best,\ 1}^1 = -9.6,\ P_{best,\ 2}^1 = -6,\ P_{best,\ 3}^1 = -2.6,\ P_{best,\ 4}^1 = -1.1,\ P_{best,\ 5}^1 = 0.6,$$

$$P_{best,\ 6}^1 = 2.3,\ P_{best,\ 7}^1 = 2.8,\ P_{best,\ 8}^1 = 8.3,\ P_{best,\ 9}^1 = 10$$

- **Step 4:** Gbest = max(Pbest) so Gbest = 2.3.
- **Step 5:** Updating the velocities of the particle by considering the value of random numbers

$r_1 = 0.213$, $r_2 = 0.876$, $C_1 = C_2 = 1$, $w = 1$.

$$v_i^{t+1} = v_i^t + C1r_1^t\left[P_{best,\, i}^t - x_i^t\right] + C2r_2^t\left[G_{best}^t - x_i^t\right]; \quad i = 1,\ 2,\ 3,\ \ldots,\ \ 9.$$

$$v_1^1 = 0 + 0.213\,(-9.6 + 9.6) + 0.876\,(2.3 + 9.6) = 10.4244$$

$$v_2^1 = 7.2708,\ v_3^1 = 4.2924,\ v_5^1 = 1.4892,\ v_6^1 = 0,\ v_7^1 = -0.4380,\ v_8^1 = 5.256,\ v_9^1 = -6.7452$$

Step 6: Update the values of position as well.

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

$$x_1^1 = 0.8244,\ x_2^1 = 1.2708,\ x_3^1 = 1.6924,\ x_4^1 = 1.8784,\ x_5^1 = 2.0892,\ x_6^1 = 2.3,$$
$$x_7^1 = 2.362,\ x_8^1 = 3.044,\ x_9^1 = 3.2548$$

Step 7: Finding objective function values of

$$f_1^1 = 23.4424,\ f_2^1 = 24.739,\ f_3^1 = 25.5978,\ f_4^1 = 25.8636,\ f_5^1 = 26.0812,\ f_6^1 = 26.21,$$
$$f_7^1 = 26.231,\ f_8^1 = 25.9541,\ f_9^1 = 25.6803$$

Step 8: Stopping Criteria.

If the terminal rule is satisfied, go to step 2. Otherwise stop the iteration and note the result.

SWARM INTELLIGENCE

(ANT-COLONY OPTIMIZATION)

WHAT IS A SWARM?

- A loosely structured collection of interacting agents.
  - **Agents:**
    - Individuals that belong to a group (but are not necessarily identical).
    - They contribute to and benefit from the group.
    - They can recognize, communicate, and/or interact with each other.
- The natural perception of swarms is a group of agents in motion – but that does not always have to be the case.
- A swarm is better understood if thought of as agents exhibiting a collective behavior.

SWARM INTELLIGENCE (SI)

- An artificial intelligence (AI) technique based on the collective behavior in decentralized, self-organized systems.
  - Generally made up of agents who interact with each other and the environment.
  - No centralized control structures.
  - Based on group behavior found in nature.
- "The emergent collective intelligence of groups of simple agents." (Bonabeau et al, 1999)

EXAMPLES OF SWARMS IN NATURE:

- Classic Example: **Swarm of Bees**.
- Can be extended to other **similar systems**:
  - **Ant colony**
    - **Agents: ants**
  - **Flock of birds**
    - **Agents: birds**
  - **Traffic**
    - **Agents: cars**
  - **Crowd**
    - **Agents: humans**
  - **Immune system**
    - **Agents: cells and molecules**

CHARACTERISTICS OF SWARMS

- Composed of many individuals
- Individuals are homogeneous
- Local interaction based on simple rules
- Self-organization (No centralized Control)

SWARM INTELLIGENCE (SI) - ALGORITHM

- Inspiration from swarm intelligence has led to some highly successful optimisation algorithm.

  - **Ant Colony (-based) Optimisation** – a way to solve optimisation problems based on the way that ants indirectly communicate directions to each other.

ANT COLONY OPTIMIZATION (ACO)

- The study of artificial systems modeled after the behavior of real ant colonies and are useful in solving discrete optimization problems.
- Introduced in 1992 by Marco Dorigo.
  - Originally called it the Ant System (AS).
  - Has been applied to
    - Traveling Salesman Problem (and other shortest path problems).
    - Several NP-hard Problems.
- It is a population-based metaheuristic used to find approximate solutions to difficult optimization problems.

ACO CONCEPT

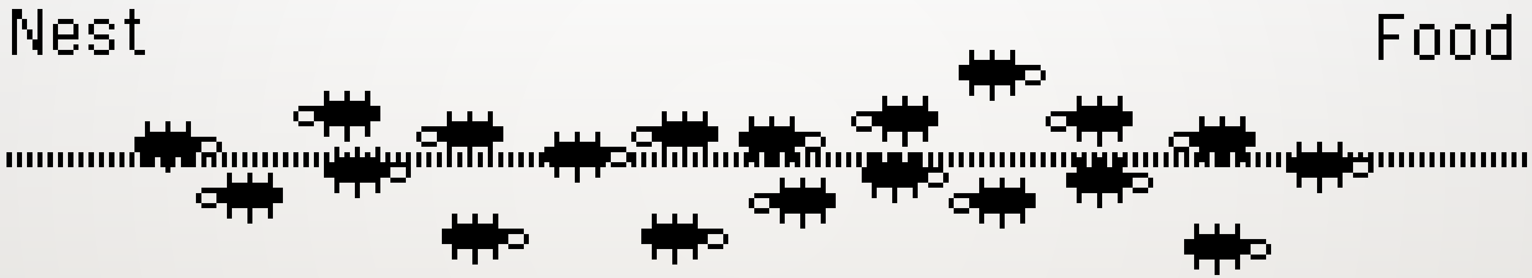
- Ant Colony Optimization (ACO) studies artificial systems that take inspiration from the *behavior of real ant colonies* and which are used to solve discrete optimization problems."
  - Ants navigate from nest to food source. Ants are blind!
  - Shortest path is discovered via pheromone trails. Each ant moves at random
  - Pheromone is deposited on path
  - More pheromone on path increases probability of path being followed

A KEY CONCEPT: STIGMERGY

- **Stigmergy** is: **indirect communication via interaction with the environment.**

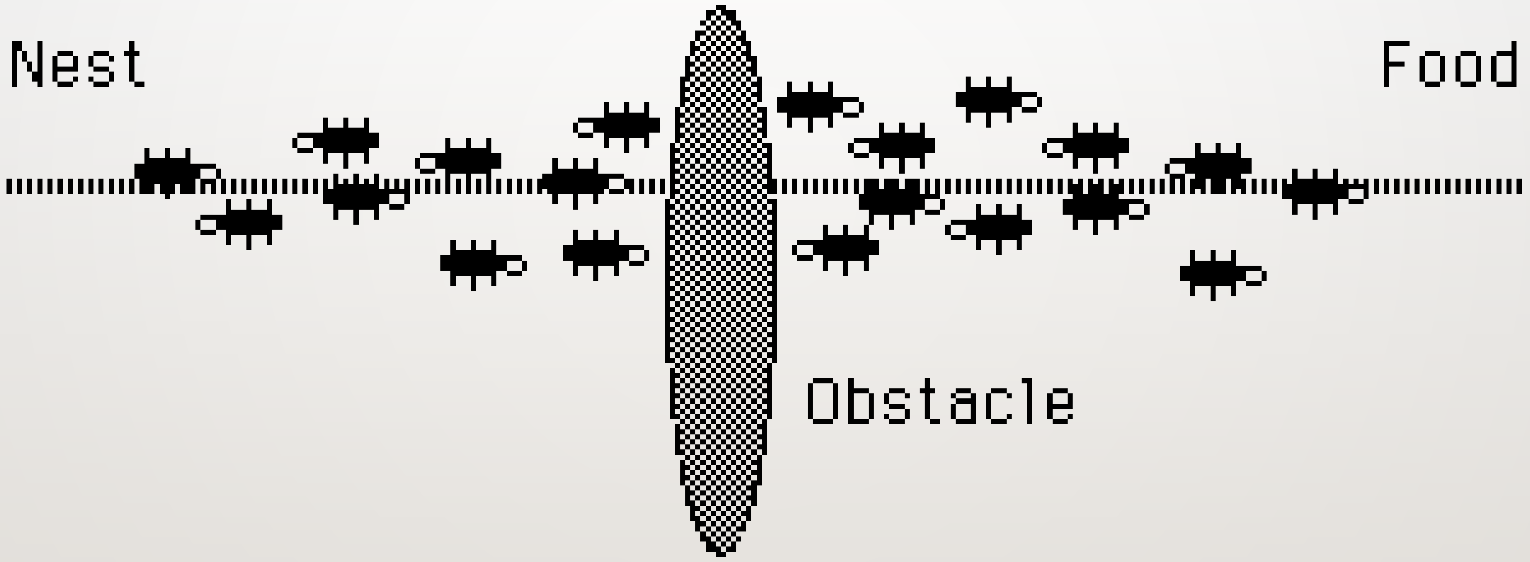
  - A problem gets solved bit by bit ..
  - Individuals communicate with each other in the above way, affecting what each other does on the task.
  - Individuals leave *markers* or *messages* – these don't solve the problem in themselves, but they affect other individuals in a way that helps them solve the problem …

NATURALLY OBSERVED ANT BEHAVIOR



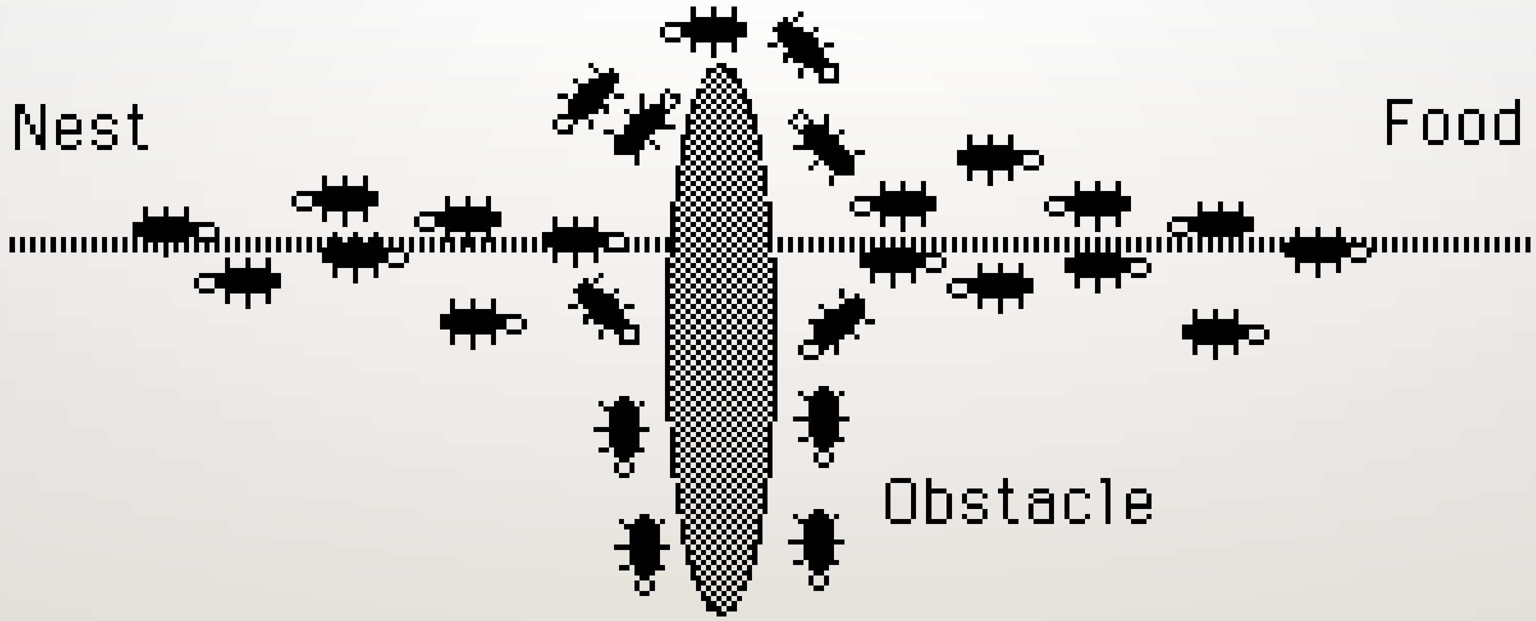
All is well in the world of the ant.

NATURALLY OBSERVED ANT BEHAVIOR



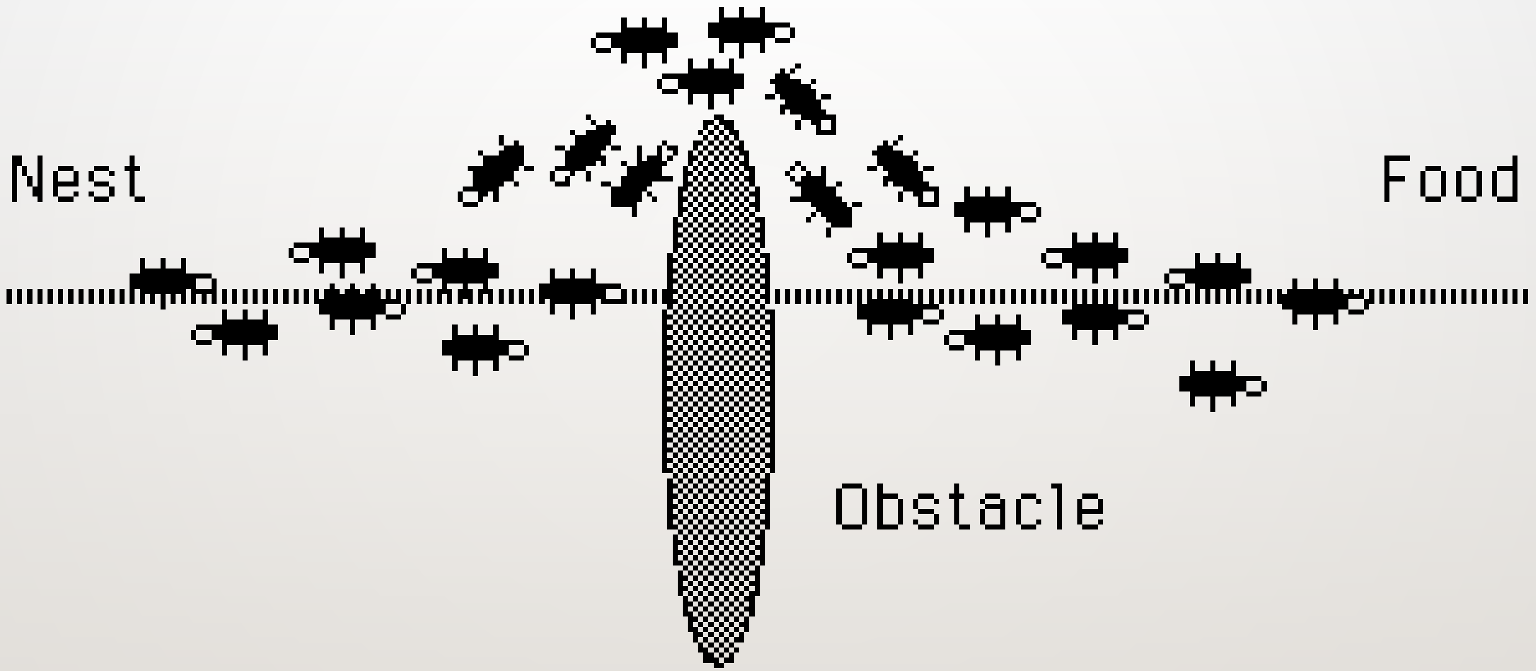
Oh no! An obstacle has blocked our path!

NATURALLY OBSERVED ANT BEHAVIOR



Where do we go? Everybody, flip a coin.

NATURALLY OBSERVED ANT BEHAVIOR



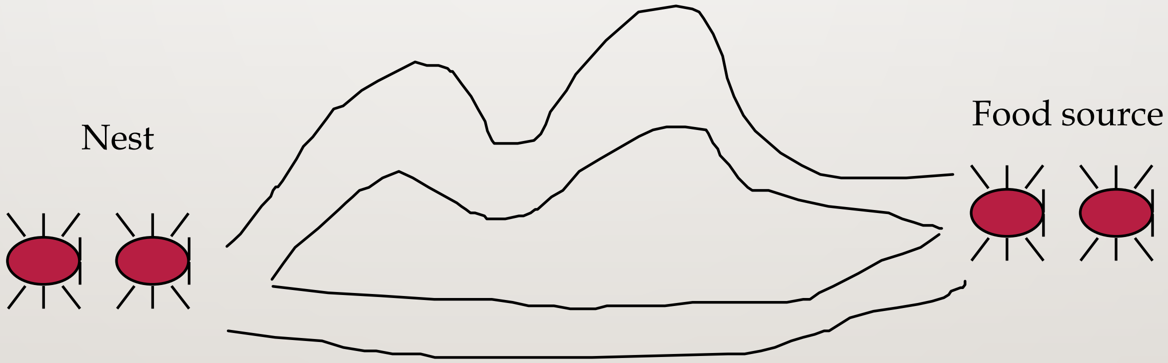
Shorter path reinforced.

STIGMERGY IN ANTS

- Ants are behaviorally unsophisticated, but collectively they can perform complex tasks.
- Ants have *highly developed sophisticated sign-based stigmergy*
  - They communicate using pheromones;
  - They **lay trails of pheromone** that can be followed by other ants.
- If an ant has a **choice of two pheromone trails** to follow, one to the NW, one to the NE, but the NW one is *stronger* – which one will it follow?

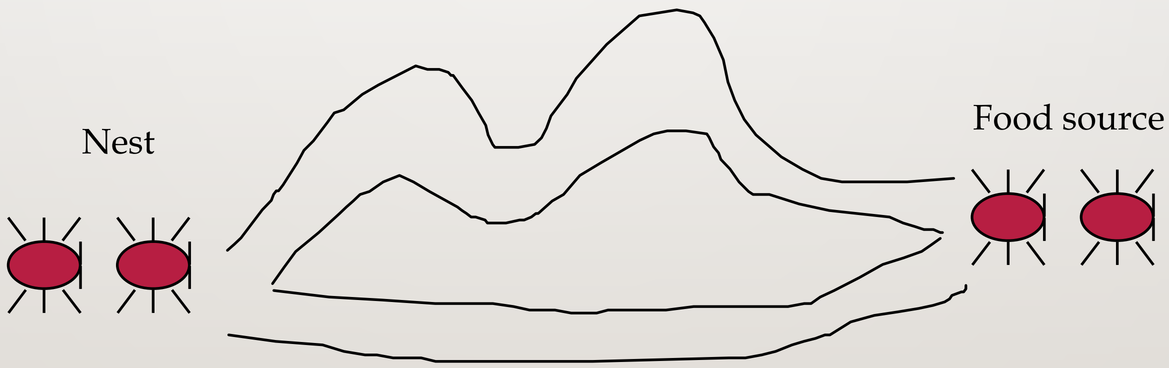
PHEROMONE TRAILS

- Individual ants lay pheromone trails while travelling from the nest, to the nest or possibly in both directions.
- The pheromone trail gradually evaporates over time.
- But pheromone trail strength accumulate with multiple ants using path.

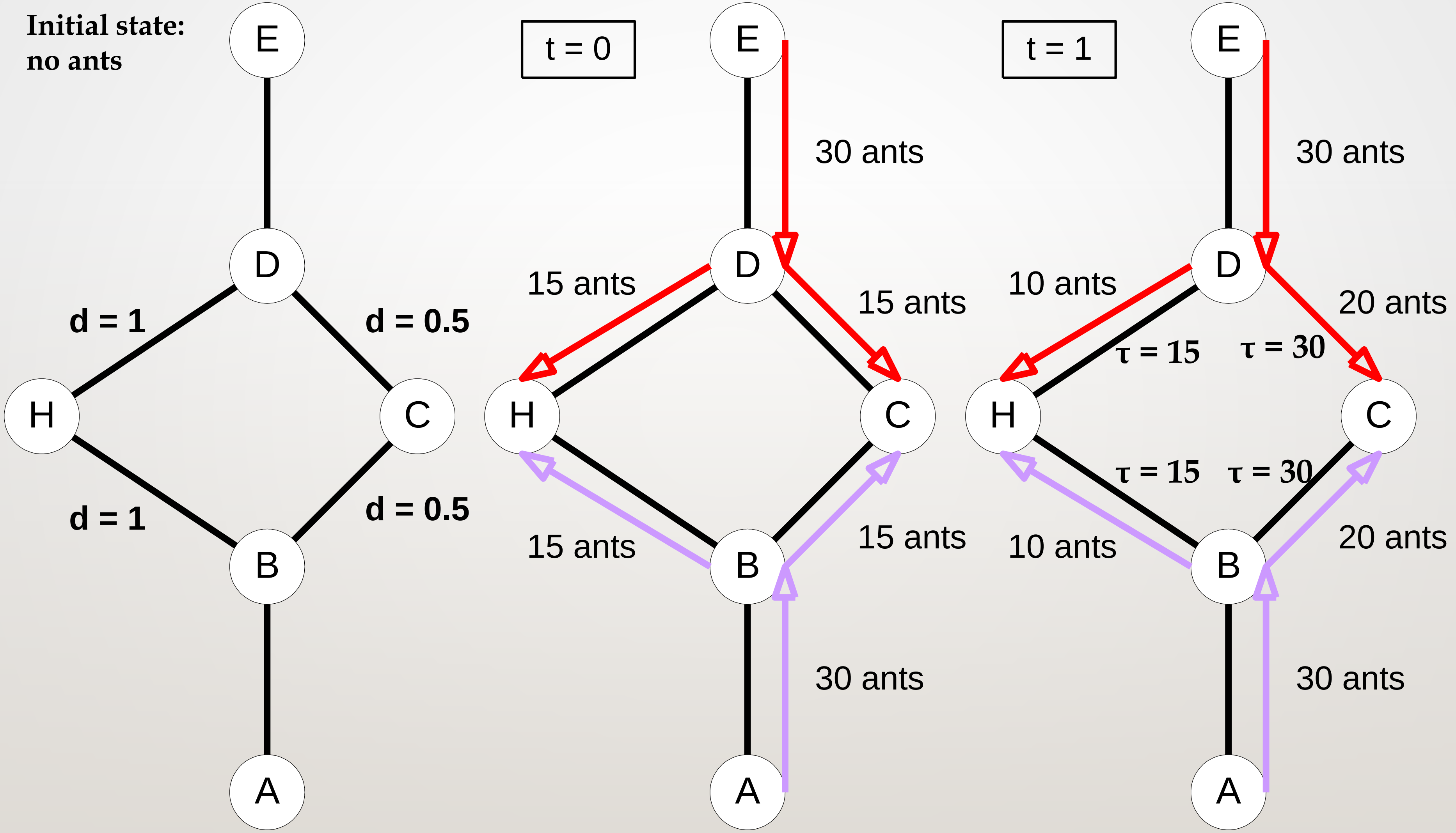


PROPERTIES OF THE PHEROMONE

- The pheromone is olfactive and volatile.
- The pheromone is stronger if more ants go along the same path(reinforced by number).
- The pheromone is stronger if the path from the nest to the food is shorter.



Pheromone Trails continued



(a) (b) (c)

ANT COLONY OPTIMISATION ALGORITHMS: BASIC IDEAS

- Ants are *agents* that:
  - Move along between nodes in a graph.
- They choose where to go based on pheromone strength.
- An ant's path represents a specific candidate solution.
- When an ant has finished a solution, pheromone is laid on its path, according to quality of solution.
- This pheromone trail affects behaviour of other ants by `stigmergy' …

USING ACO

- The optimization problem must be written in the form of a path finding problem with a weighted graph
- The artificial ants search for “good” solutions by moving on the graph
  - Ants can also build infeasible solutions – which could be helpful in solving some optimization problems
- The meta heuristic is constructed using three procedures:
  - Construct Ants Solutions
  - Update Pheromones
  - Daemon Actions

CONSTRUCT ANTS SOLUTIONS

- Manages the colony of ants.
- Ants move to neighboring nodes of the graph.
- Moves are determined by stochastic local decision policies based on pheromone trails and heuristic information.
- Evaluates the current partial solution to determine the quantity of pheromones the ants should deposit at a given node.

UPDATE PHEROMONES

- Process for modifying the pheromone trails
- Modified by
  - Increase
    - Ants deposit pheromones on the nodes (or the edges)
  - Decrease
    - Ants don't replace the pheromones and they evaporate
- Increasing the pheromones increases the probability of paths being used (i.e., building the solution)
- Decreasing the pheromones decreases the probability of the paths being used (i.e., forgetting)

DAEMON ACTIONS

- Used to implement larger actions that require more than one ant
- Examples:
  - Perform a local search
  - Collection of global information

A GENERAL ALGORITHM

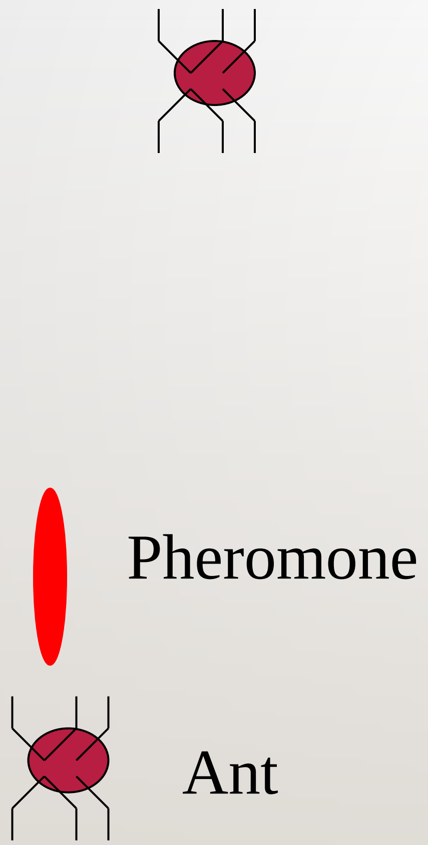
- Step1: Initialize the pheromone information.
- Step 2 : for each ant, do the following:
  - Find a solution (a path) based on the current pheromone trail.
  - Reinforcement : add pheromone.
  - Evaporation: reduce pheromone.
- Step 3 : stop if terminating condition satisfied, return to step 2 other wise.

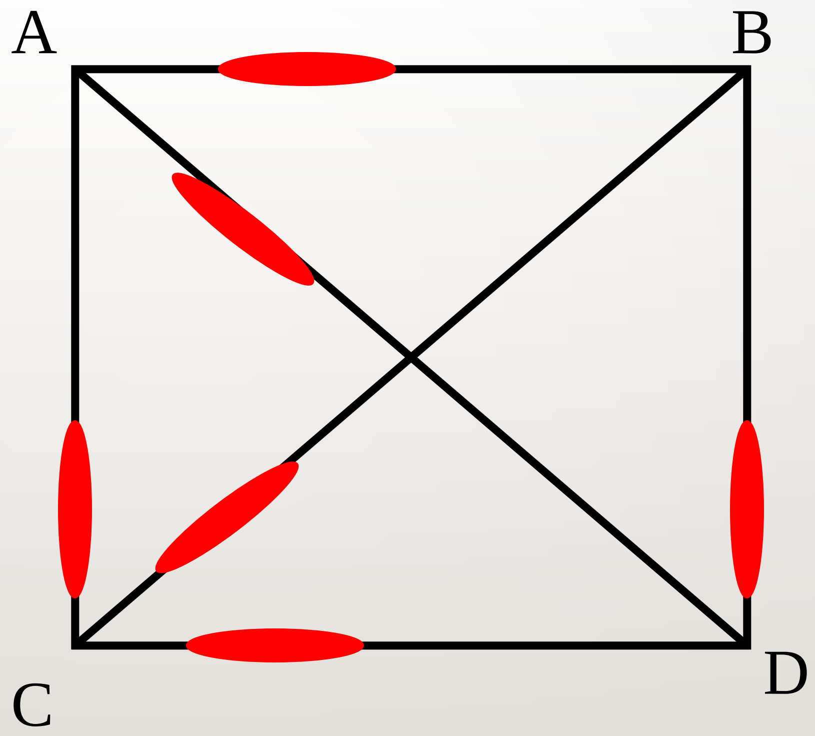
APPLICATIONS OF ACO

- Vehicle routing with time window constraints
- Network routing problems
- Assembly line balancing
- Heating oil distribution
- Data mining
- Robotic Path Problem

E.G. A 4-CITY TSP

Initially, random levels of pheromone are scattered on the edges

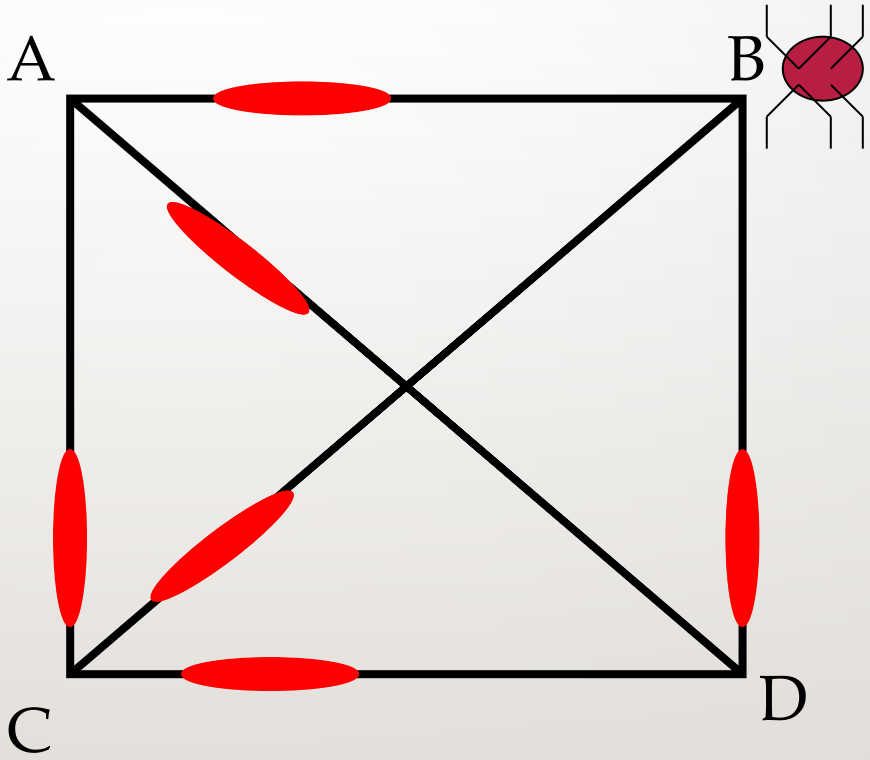


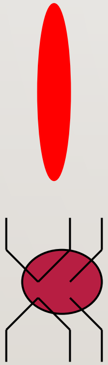


AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

An ant is placed at a random node



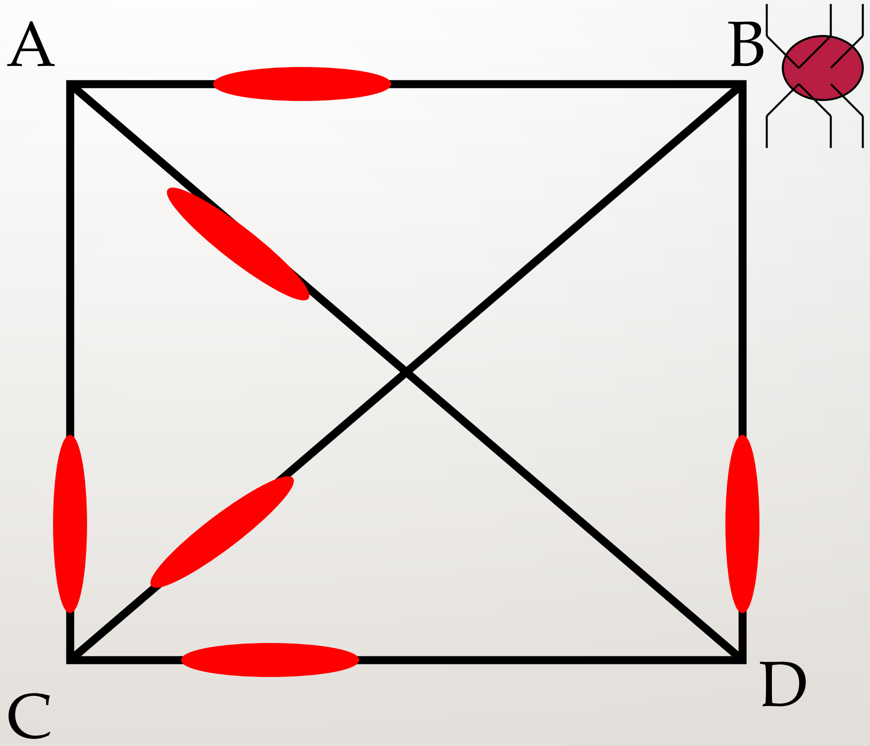


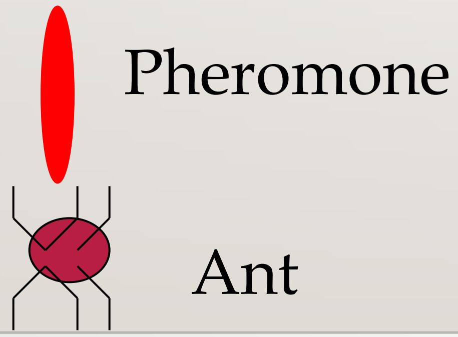
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

The ant decides where to go from that node, based on probabilities calculated from:
- pheromone strengths,
- next-hop distances.

Suppose this one chooses BC





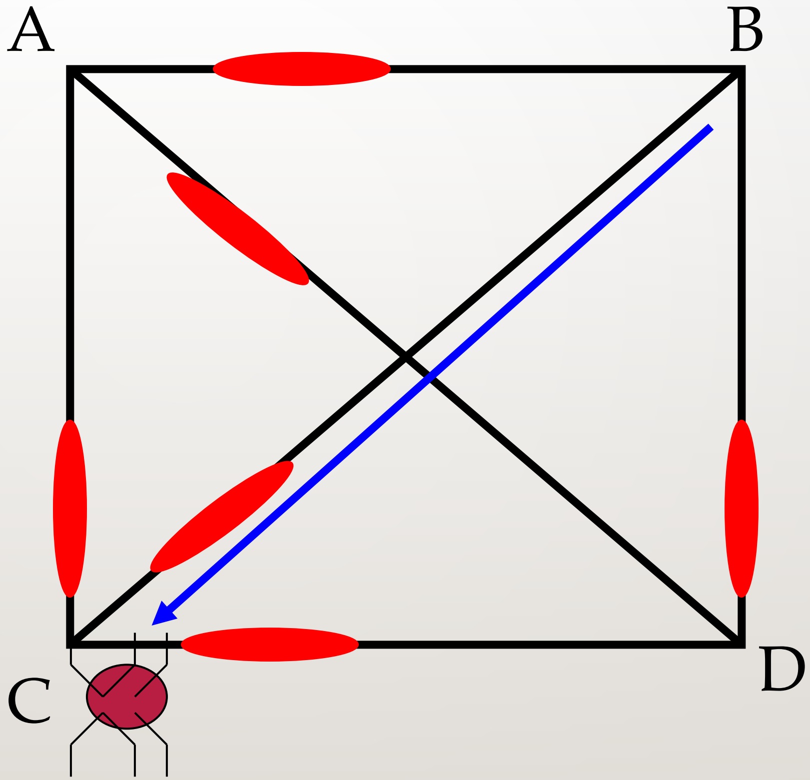
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

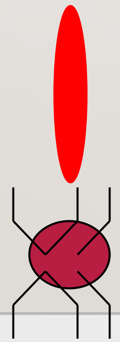
The ant is now at C, and has a `tour memory' = {B, C} – so he cannot visit B or C again.

Again, he decides next hop (from those allowed) based on pheromone strength and distance;

suppose he chooses CD



AB: 10, AC: 10, AD, 30, BC, 40, CD 20



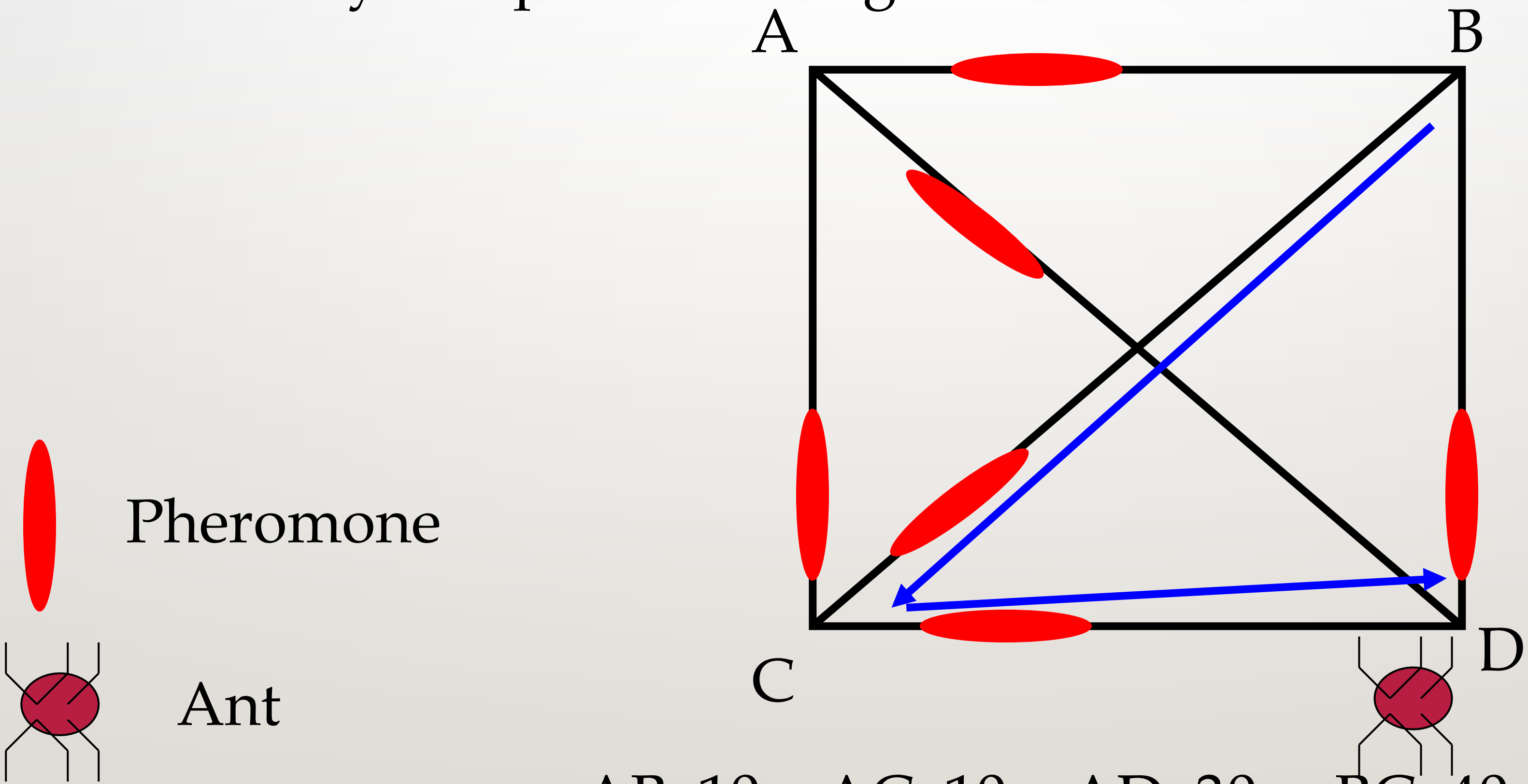
Pheromone

Ant

E.G. A 4-CITY TSP

The ant is now at D, and has a `tour memory' = {B, C, D}

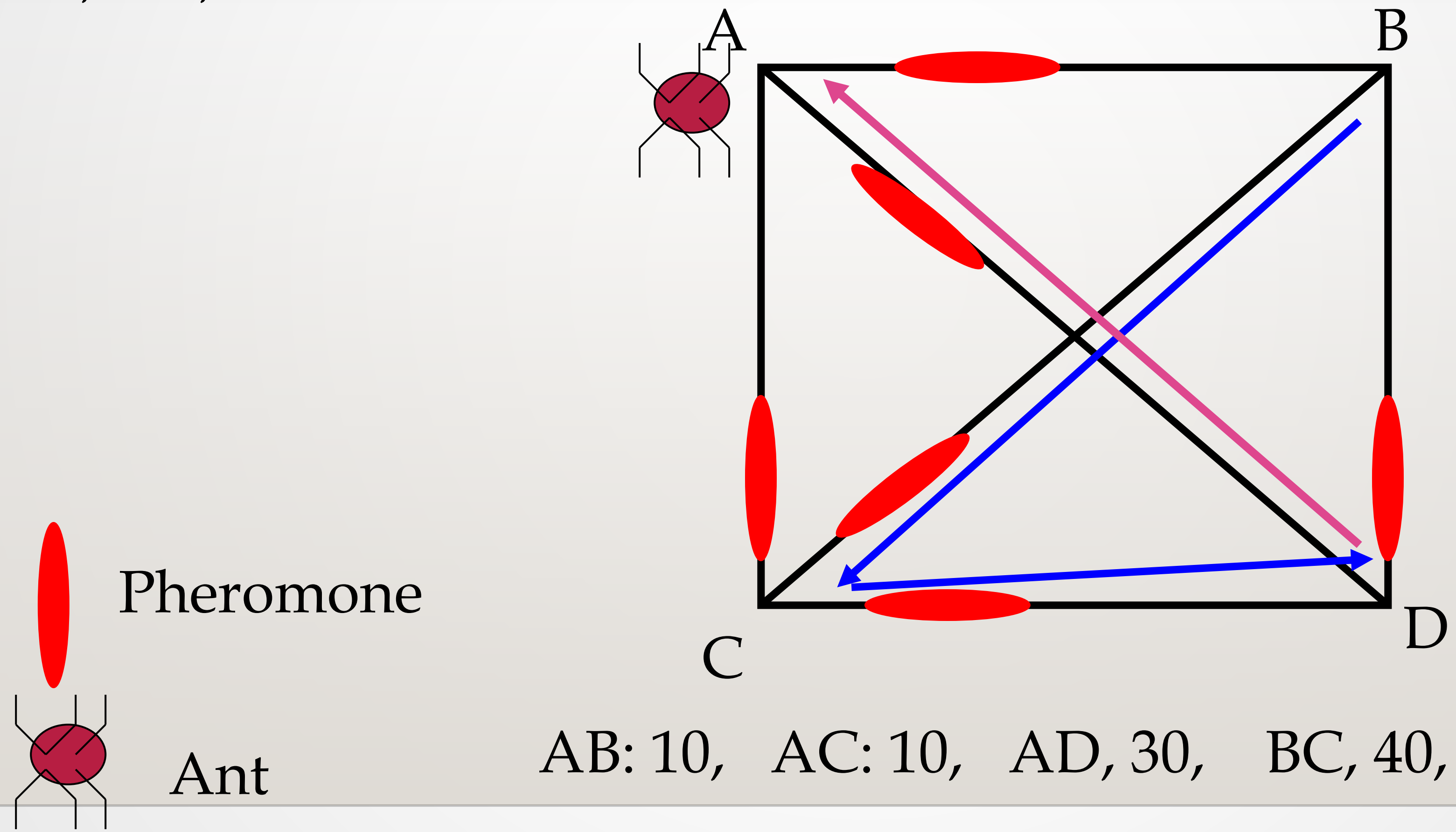
There is only one place he can go now:



AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

So, he has nearly finished his tour, having gone over the links: BC, CD, and DA.

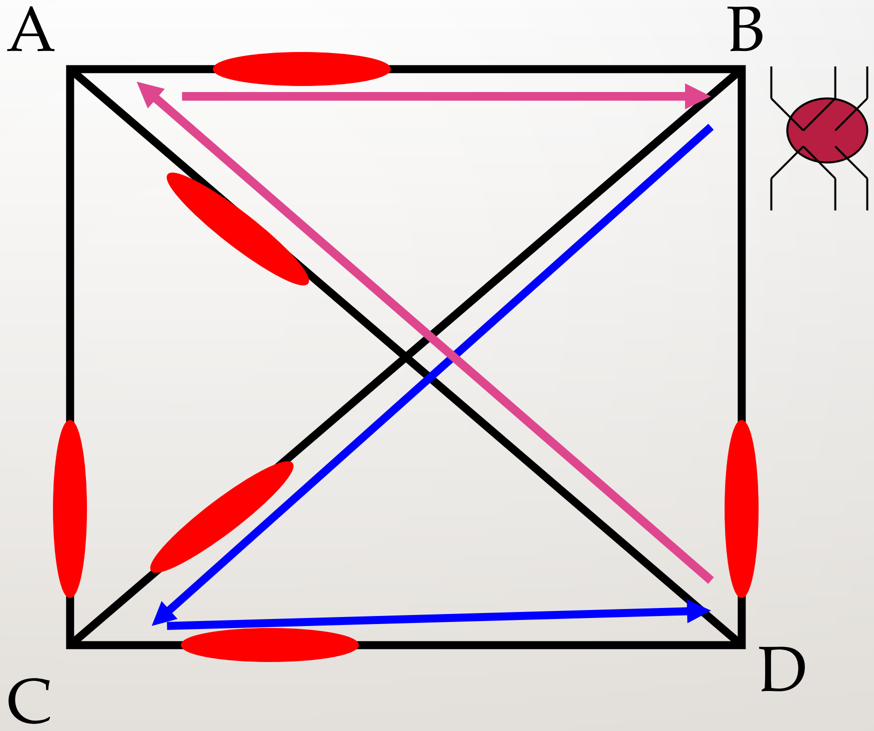


AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

So, he has nearly finished his tour, having gone over the links: BC, CD, and DA. AB is added to complete the round trip.

Now, pheromone on the tour is increased, in line with the fitness of that tour.



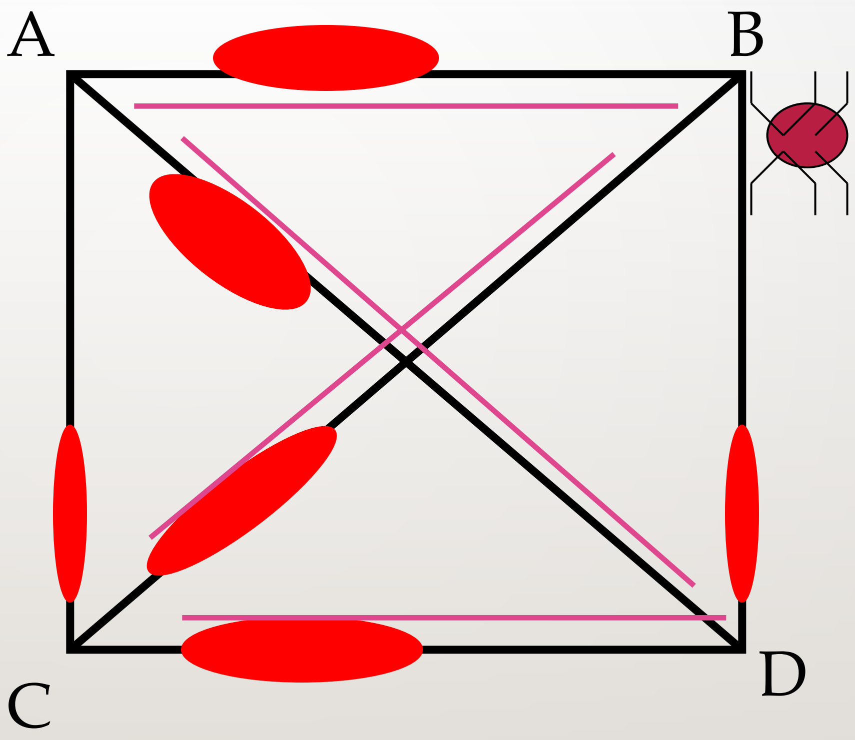
Pheromone

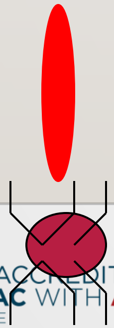
Ant

AB: 10, AC: 10, AD, 30, BC, 40, CD 20

E.G. A 4-CITY TSP

Next, pheromone everywhere is decreased a little, to model decay of trail strength over time





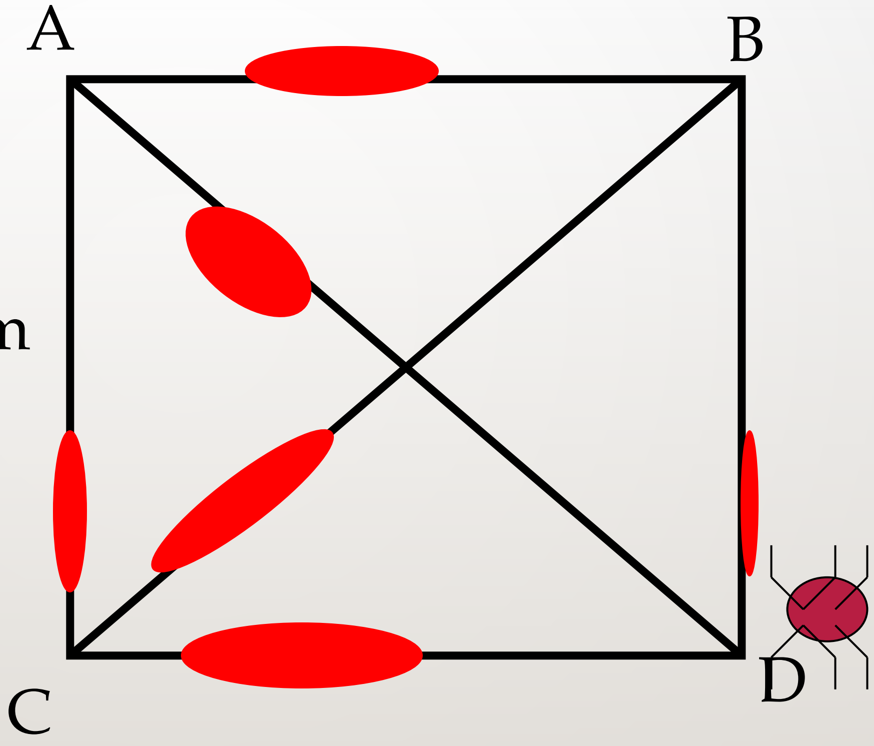
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

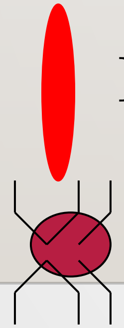
E.G. A 4-CITY TSP

We start again, with another ant in a random position.

Where will he go?

Next , the actual algorithm and variants.





Pheromone

Ant

AB: 10, AC: 10, AD, 30, BC, 40, CD 20

THE ACO ALGORITHM FOR THE TSP

We have a TSP, with *n* cities.

1. We place some ants at each city. Each ant then does this:
   - It makes a complete tour of the cities, coming back to its starting city, using a *transition rule* to decide which links to follow. By this rule, it chooses each next-city at random, but based partly by the pheromone levels existing at each path, and based partly by *heuristic information.*

2. When all ants have completed their tours.

   Global Pheromone Updating occurs.
   - The current pheromone levels on all links are reduced (I.e. pheromone levels decay over time).
   - Pheromone is laid (belatedly) by each ant as follows: it places pheromone on all links of its tour, with strength depending on how good the tour was.

THE ACO ALGORITHM FOR THE TSP [A SIMPLIFIED VERSION WITH ALL ESSENTIAL DETAILS]

We have a TSP, with *n* cities.

1. We place some ants at each city. Each ant then does this:
   - It makes a complete tour of the cities, coming back to its starting city, using a *transition rule* to decide which links to follow. By this rule, it chooses each next-city at random, but biased partly by the pheromone levels existing at each path, and biased partly by *heuristic information.*
2. When all ants have completed their tours.

   Global Pheromone Updating occurs.
   - The current pheromone levels on all links are reduced (I.e. pheromone levels decay over time).
   - Pheromone is lain (belatedly) by each ant as follows: it places pheromone on all links of its tour, with strength depending on how good the tour was.

Then we go back to 1 and repeat the whole process many times, until we reach a termination criterion.

ACO Algorithm

Set all parameters and initialize the pheromone trails

Loop

Sub-Loop

Construct solutions based on the state transition rule

Apply the online pheromone update rule

Continue until all ants have been generated

Apply Local Search

Evaluate all solutions and record the best solution so far

Apply the offline pheromone update rule

Continue until the stopping criterion is reached

THE TRANSITION RULE

T(*r*,s) is the amount of pheromone currently on the path that goes directly from city *r* to city *s*.

H(*r*,*s*) is the heuristic value of this link – in the classic TSP application, this is chosen to be 1/distance(*r*,*s*) -- i.e. the shorter the distance, the higher the heuristic value.

$p_k(r,s)$ is the probability that ant *k* will choose the link that goes from *r* to *s*

β is a parameter that we can call the *heuristic strength*

The rule is:

$$p_k(r,s) = \frac{T(r,s) \cdot H(r,s)^{\beta}}{\sum_{\text{unvisited cities } c} T(r,c) \cdot H(r,c)^{\beta}}$$

Where our ant is at city *r*, and *s* is a city as yet unvisited on its tour, and the summation is over all of *k*'s unvisited cities

GLOBAL PHEROMONE UPDATE

$A_k(r,s)$	is amount of pheromone added to the (r, s) link by ant k.
m	is the number of ants
ρ	is a parameter called the pheromone decay rate.
L_k	is the length of the tour completed by ant k

$T(r, s)$ at the next iteration becomes:

$$\rho \cdot T(r,s) + \sum_{k=1}^{m} A_k(r,s)$$

Where $A_k(r,s) = 1/L_k$

Ant Colony Optimization

Characteristics

- An ant is a solution.
- Solutions (ants) are at different places in the solution space.
- How they change is based on the probability of changing to a different schedule.
- An ant completes its tour after selection a choice for each stand.
- Utilities (objective function values) of each tour are calculated.
- Pheromone levels are updated after all of the ants have completed all of their tours.

Ant Colony Optimization

Advantages:

- It is intuitive to biologically-minded people, mimicking nature.

- The system is built on positive feedback (pheromone attraction) and negative attractiveness (pheromone evaporation).

- Pheromone evaporation helps avoid convergence to a local optima.

Disadvantages:

- For routing problems it may make more sense, but for harvest scheduling problems, it requires a conceptual leap of faith.

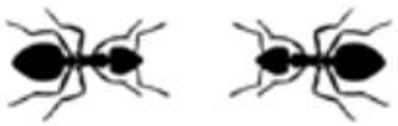
- Fine-tuning the sensitive parameters may require significant effort.

Example

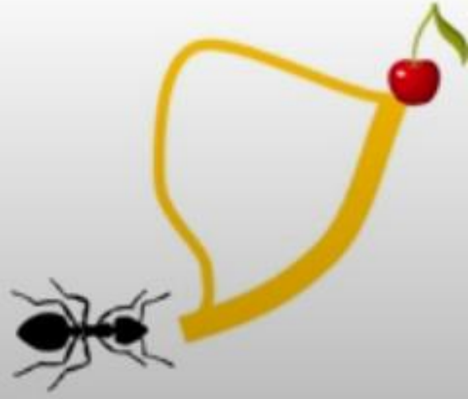
An ANT is at a distance of 5m from the TREE 15m from CAR and 4m from a DOLL, the distance between TREE and the CAR is 4m,CAR and DOLL is 1m ,DOLL and TREE is 8m.Create a matrix and solve using Ant Colony Optimization.

- Ants produces chemicals called **pheromone**

- They use **pheromone** to communicate. This is similar to **water in our analogy**.



- They use a very similar technique as our analogy to find the shortest path from their nest to a source of food.



ANT COLONY OPTIMIZATION

- Ant Colony Optimization(ACO) is a nature-inspired metaheuristic algorithm that simulates the foraging behaviour of ants to find optimal solutions to complex problems.
- Initially proposed by Marco Dorigo in 1992.
- Aims to search for an optimal path in a graph, based on the behaviour of ants seeking path between their colony and a source of food.

ACO CONCEPT

ACO is inspired by the foraging behaviour of ants, where they find the shortest path to food sources using pheromone communication.

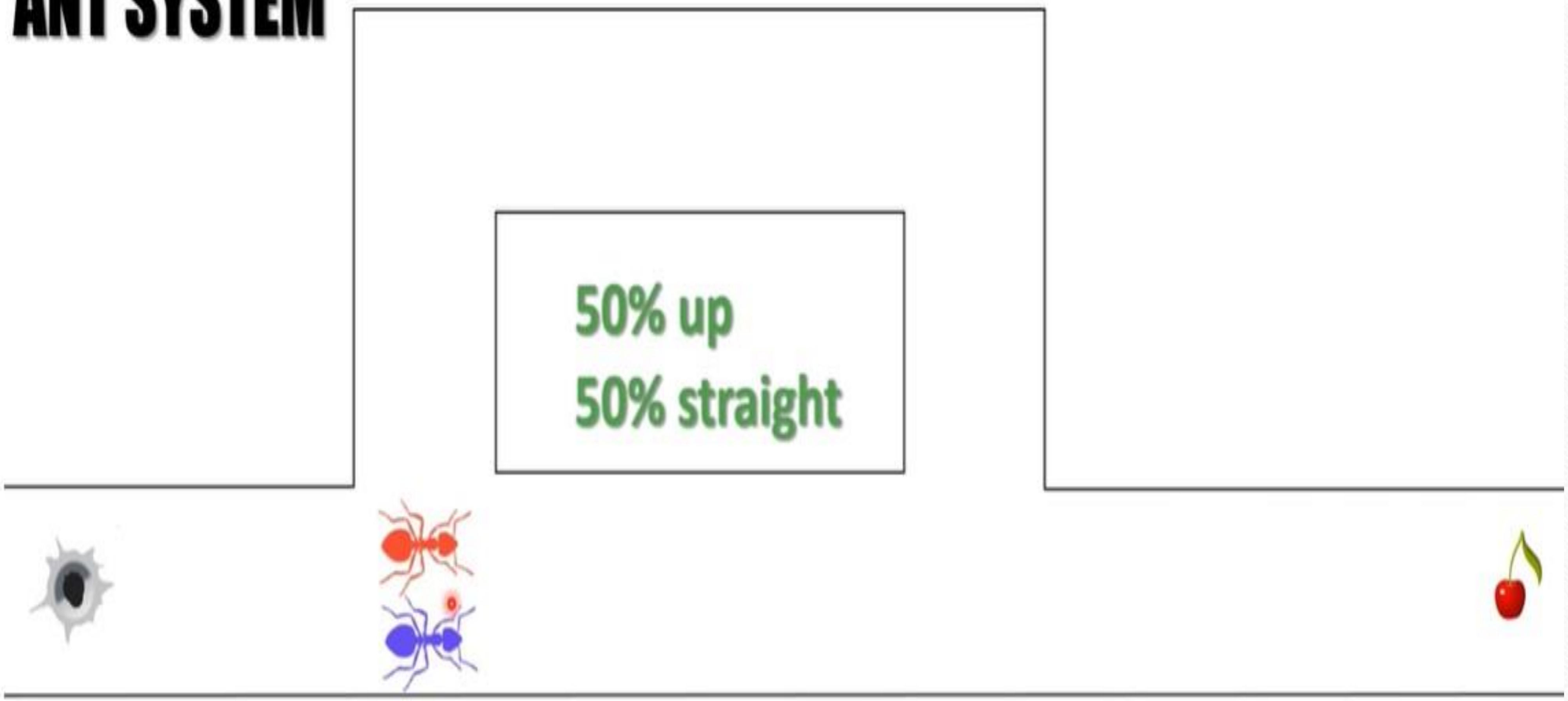
The first version of ACO was called Ant Systems.

ACO applied to the Travelling Salesman Problem, demonstrating improved solutions over traditional algorithms.

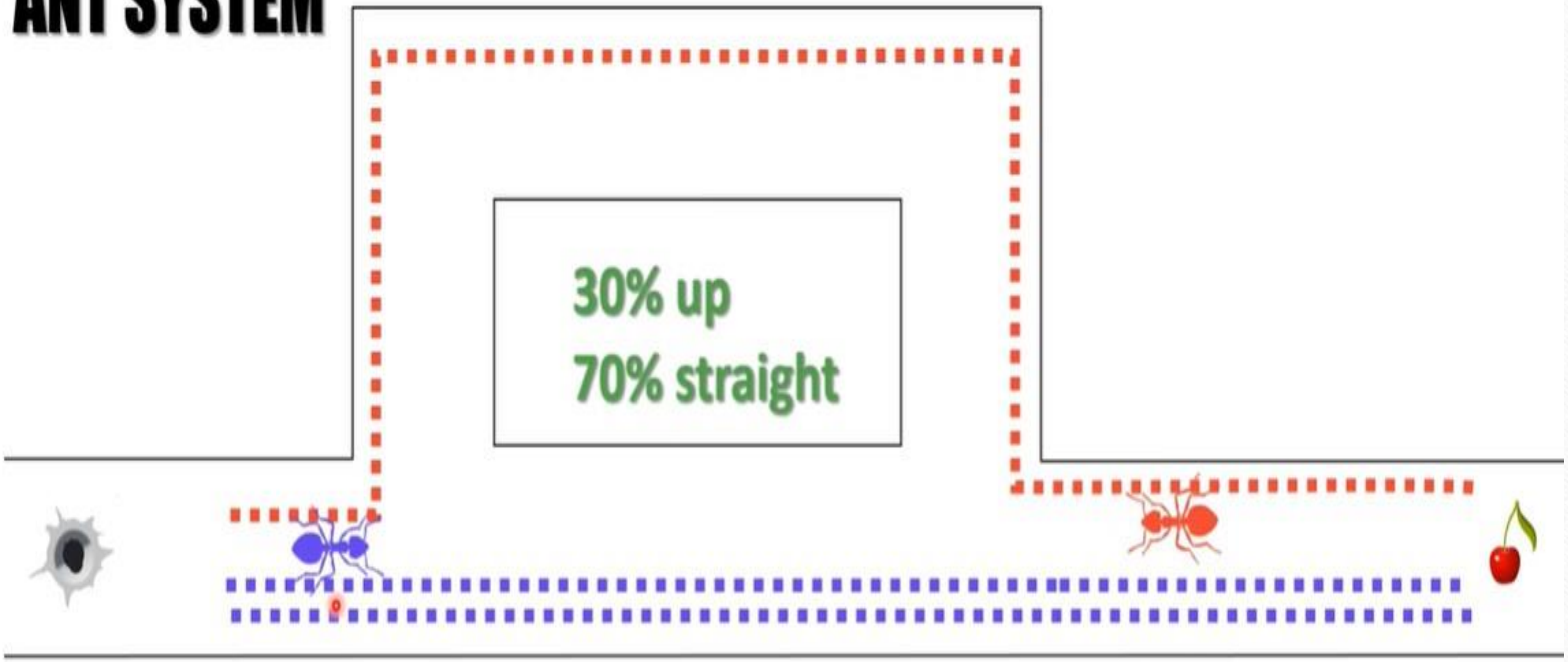
STIGMERGY

- The main inspiration of the ACO algorithm comes from stigmergy.
- Stigmergy refers to the interaction and coordination of organisms in nature by modifying the environment.
- Ants produces chemicals called pheromone. They use pheromone to communicate.
- Ants are more likely to choose path with higher pheromone.

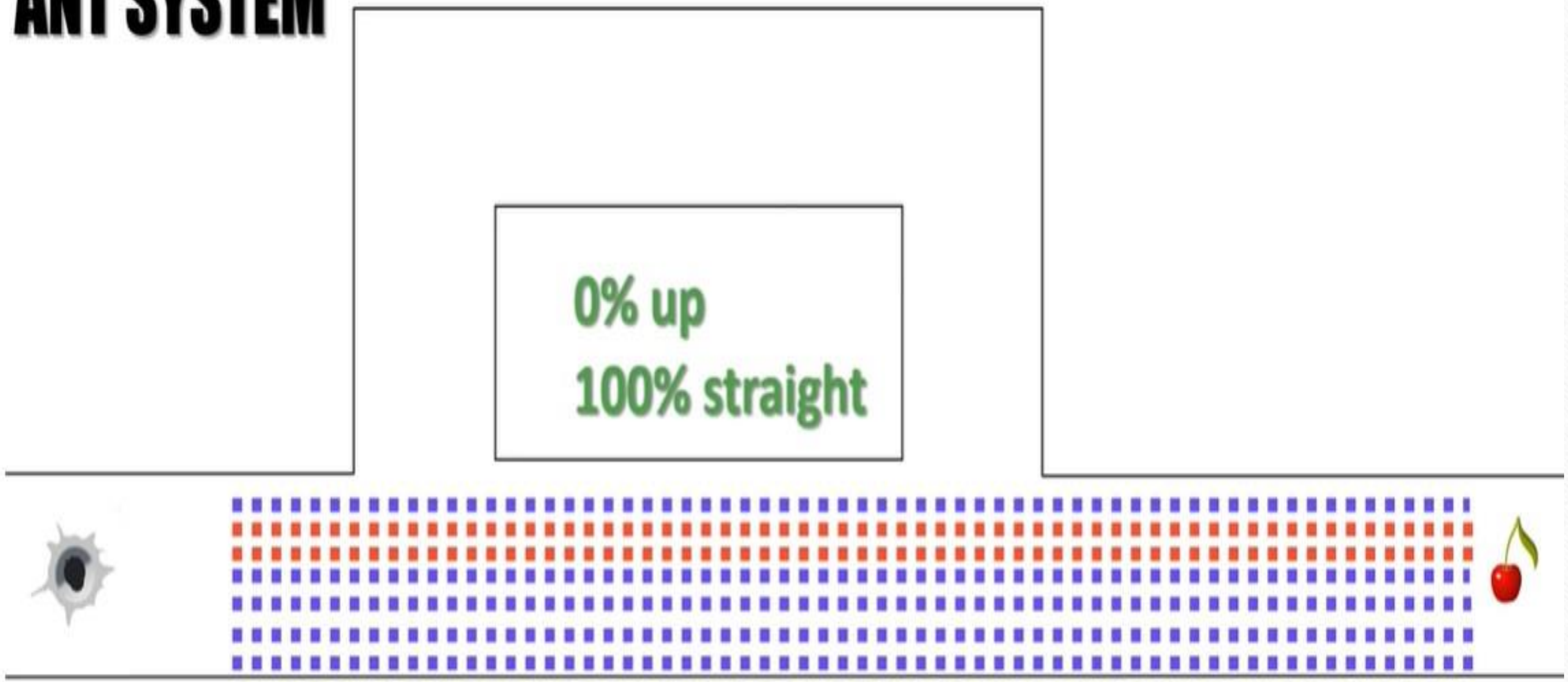
ANT SYSTEM



ANT SYSTEM



ANT SYSTEM



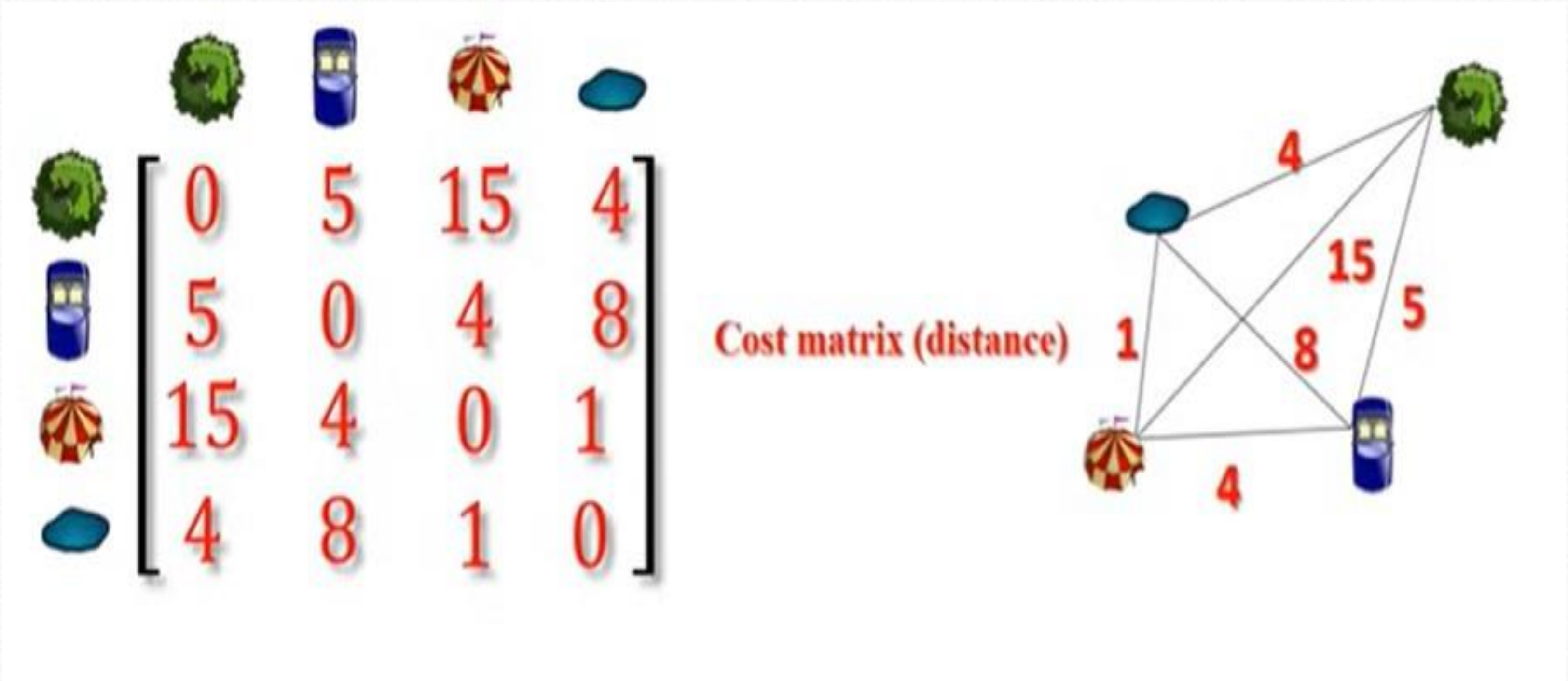
GENERAL ALGORITHM

- **Ant Movement:** Place artificial ants randomly and let them move around the problem space.
- **Pheromone Update:** Ants leave pheromone on their paths based on solution quality.
- **Solution Evaluation:** Evaluate the quality of solutions based on an objective function.
- **Repeat and Improve:** Keep repeating steps 1-3, allowing ants to iteratively improve their paths.

NUMERICAL EXAMPLE

- **An ant is at a distance of 5m from the Tree, 15m from CAR and 4m from a POND, the distance between TREE and CAR is 4m, CAR and POND is 1m, POND and TREE is 8m. Create a matrix and solve using Ant Colony Optimization.**

Cost Matrix



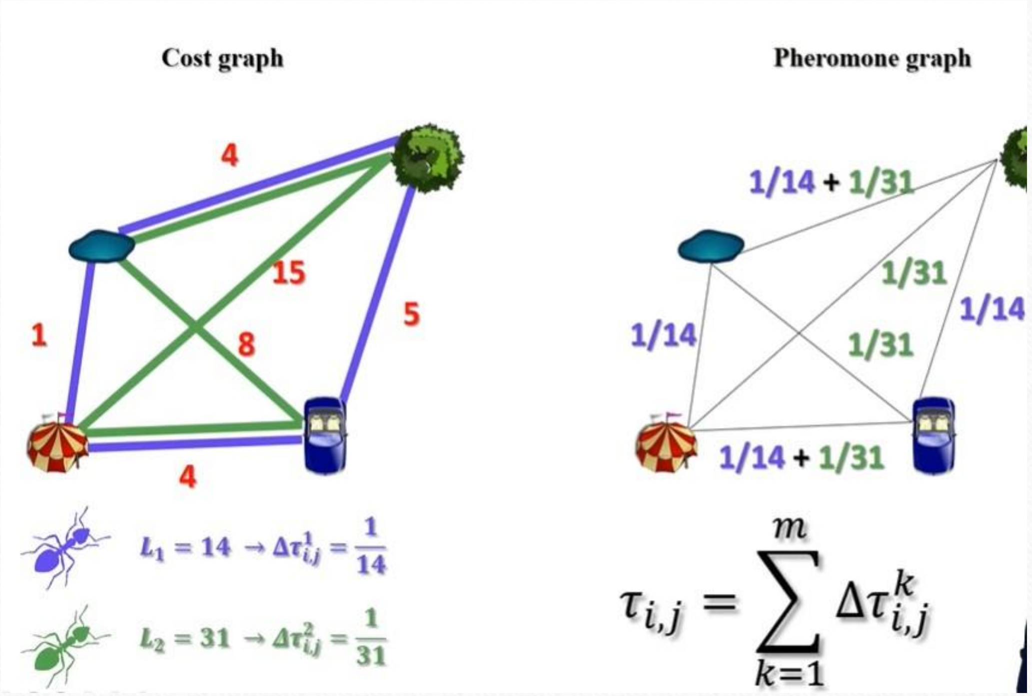
MATHEMATICAL MODEL

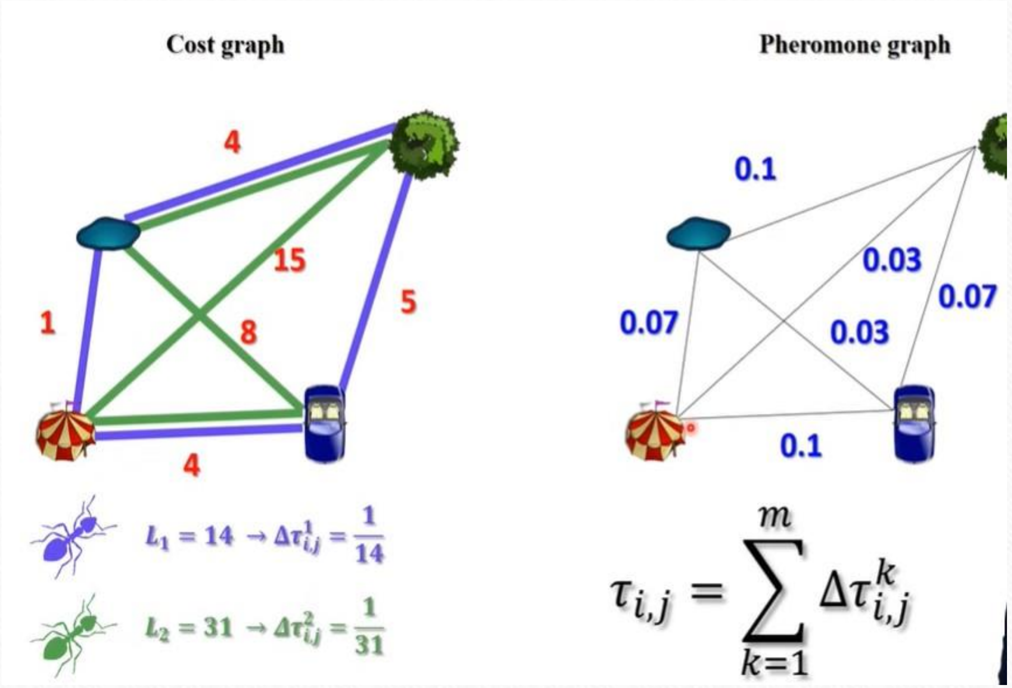
$$\Delta\tau_{i,j}^{k} = \begin{cases} \dfrac{1}{L_k} & k^{th} \text{ant travels on the edge } i,j \\ 0 & \text{otherwise} \end{cases}$$

$$\tau_{i,j}^{k} = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$ Without vaporization

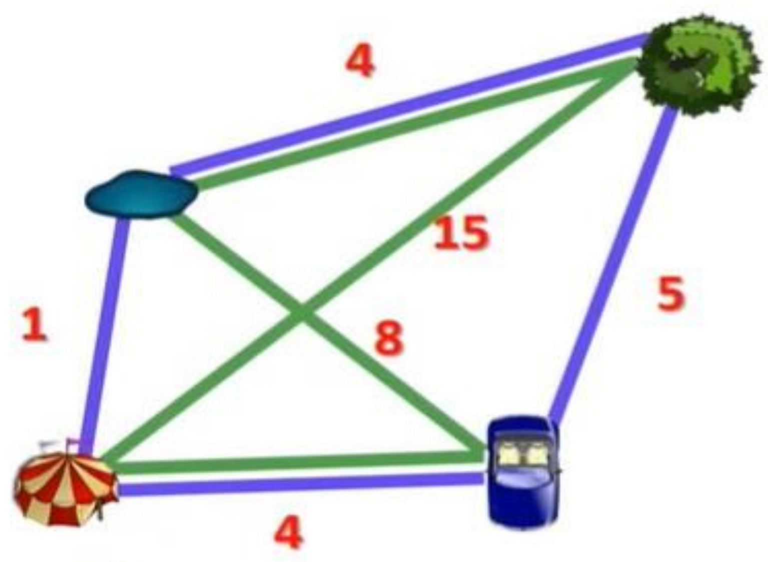
$$\tau_{i,j}^{k} = (1-\rho)\,\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$ With vaporization

Cost And Pheromone Graph





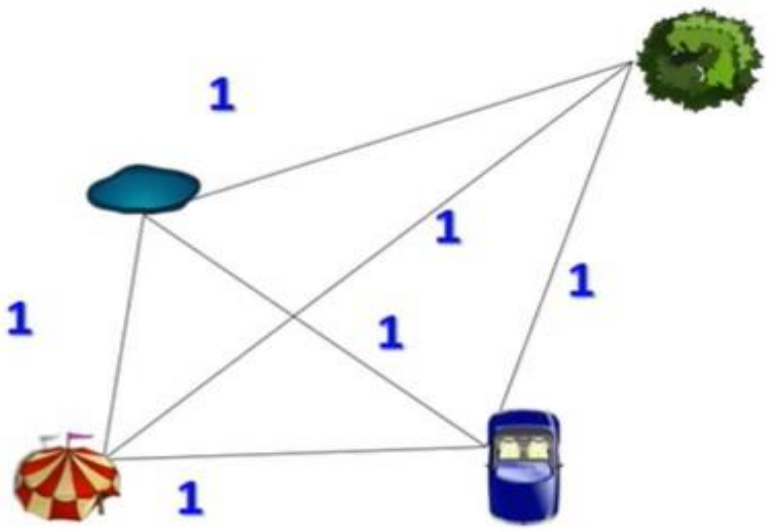
Cost graph



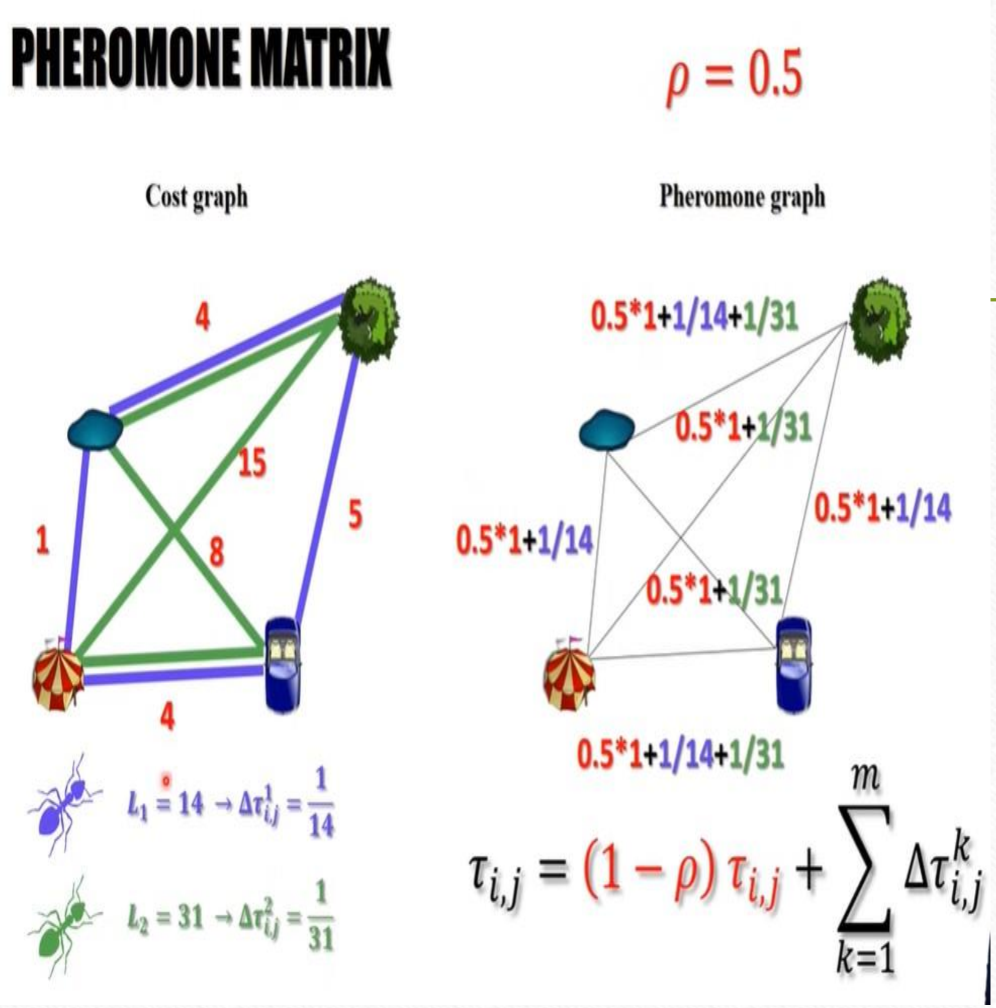
$L_1 = 14 \rightarrow \Delta\tau^1_{i,j} = \frac{1}{14}$

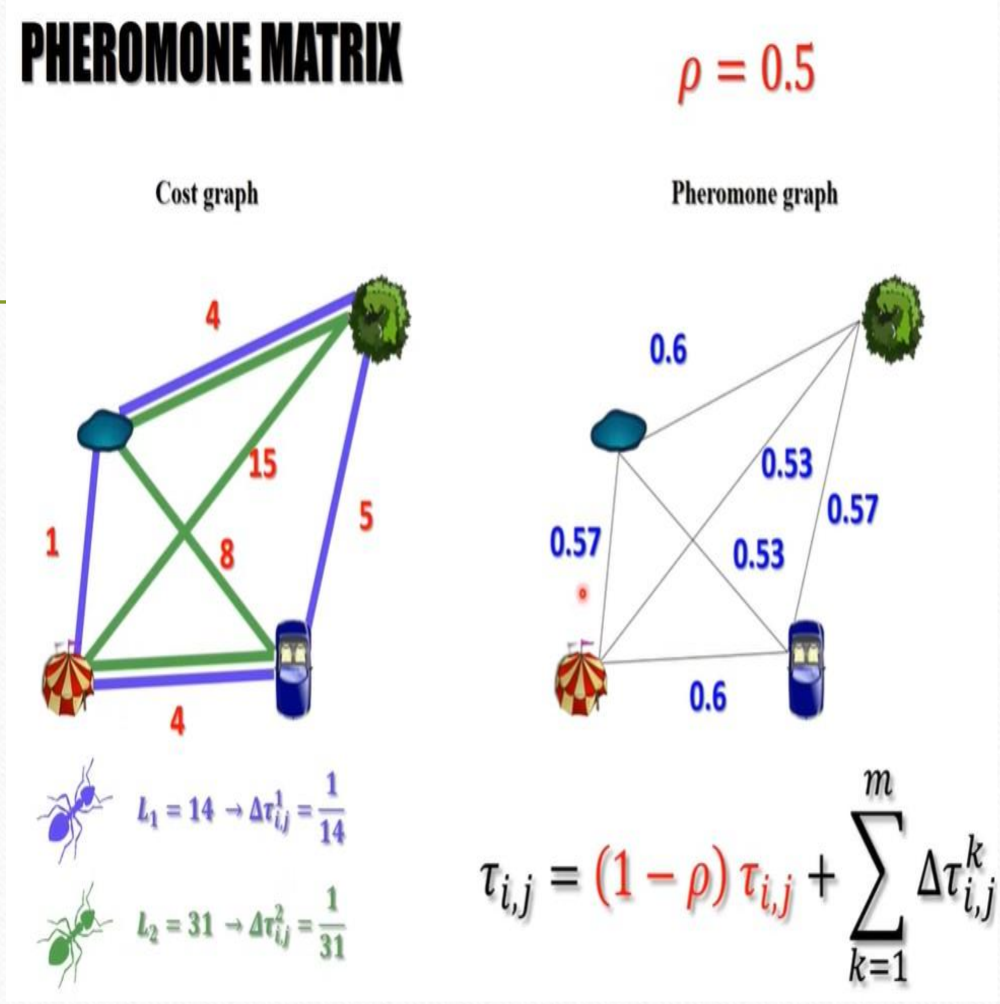
$L_2 = 31 \rightarrow \Delta\tau^2_{i,j} = \frac{1}{31}$

Pheromone graph



$$\tau_{i,j} = (1-\rho)\,\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau^k_{i,j}$$


PHEROMONE MATRIX
$\rho = 0.5$
Cost graph
Pheromone graph
4
15
1
8
5
4
0.5*1+1/14+1/31
0.5*1+1/31
0.5*1+1/14
0.5*1+1/14
0.5*1+1/31
0.5*1+1/14+1/31
$L_1 = 14 \rightarrow \Delta\tau_{i,j}^{1} = \frac{1}{14}$
$L_2 = 31 \rightarrow \Delta\tau_{i,j}^{2} = \frac{1}{31}$
$\tau_{i,j} = (1-\rho)\,\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$

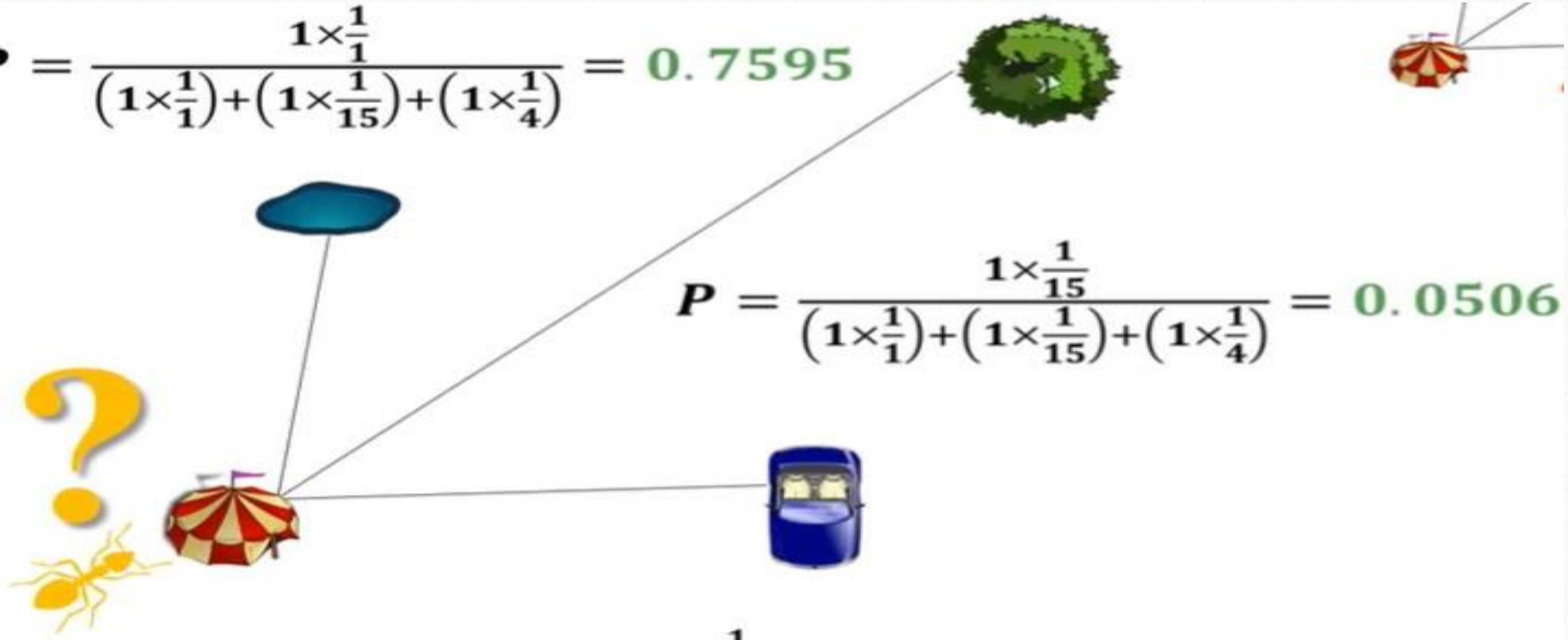

PHEROMONE MATRIX
$\rho = 0.5$
Cost graph
Pheromone graph
4
15
1
8
5
4
0.6
0.53
0.57
0.57
0.53
0.6
$L_1 = 14 \rightarrow \Delta\tau_{i,j}^{1} = \frac{1}{14}$
$L_2 = 31 \rightarrow \Delta\tau_{i,j}^{2} = \frac{1}{31}$
$\tau_{i,j} = (1-\rho)\,\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$

CALCULATING THE PROBABILITIES

$$P_{i,j} = \frac{\left(\tau_{i,j}\right)^{\alpha}\left(\eta_{i,j}\right)^{\beta}}{\sum\left(\left(\tau_{i,j}\right)^{\alpha}\left(\eta_{i,j}\right)^{\beta}\right)}$$

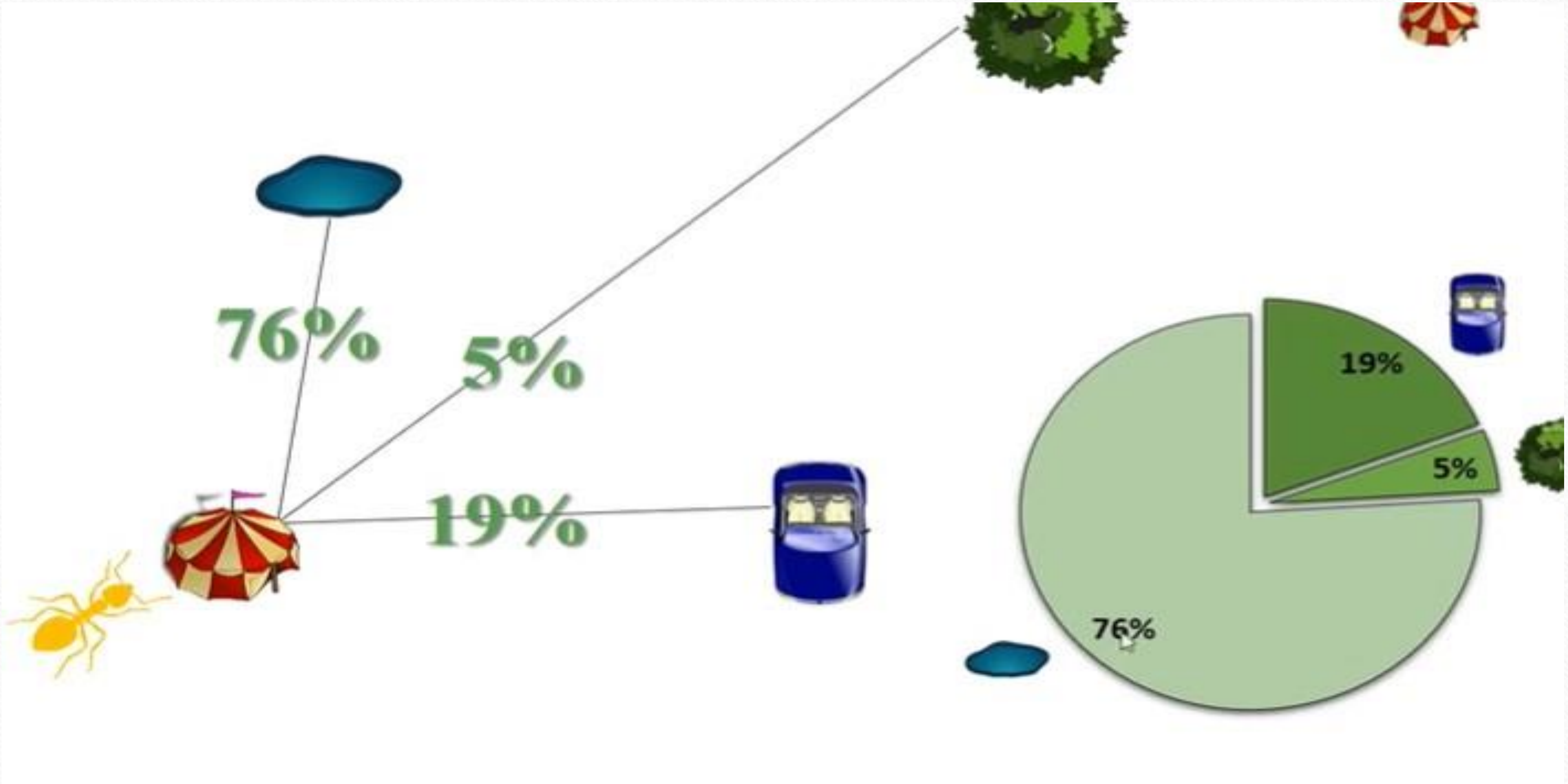
$$where\colon\ \eta_{i,j} = \frac{1}{\mathrm{L}_{i,j}}$$

$$P = \frac{1\times\frac{1}{1}}{\left(1\times\frac{1}{1}\right)+\left(1\times\frac{1}{15}\right)+\left(1\times\frac{1}{4}\right)} = 0.7595$$



$$P = \frac{1\times\frac{1}{15}}{\left(1\times\frac{1}{1}\right)+\left(1\times\frac{1}{15}\right)+\left(1\times\frac{1}{4}\right)} = 0.0506$$

$$P = \frac{1\times\frac{1}{4}}{\left(1\times\frac{1}{1}\right)+\left(1\times\frac{1}{15}\right)+\left(1\times\frac{1}{4}\right)} = 0.1899$$


76%
5%
19%
19%
5%
76%

SIMULATED ANNEALING

Overview of Simulated Annealing

- Random search method
- Developed by Kirkpatrick et al. in 1983
- Analogy with Physical Annealing Process
- Compared to hill climbing, Simulated Annealing allows downward steps or transition to weaker solutions as well
- The process generates random moves

Simulated Annealing

- Simulated Annealing is a variant of the hill climbing method
- However, it allows downhill moves as well
- Additionally, it explores the search space in such a manner that the starting point does not influence much the final solution

Simulated Annealing

- Inspired by the Annealing process in which materials are raised to high energy levels for melting and are then cooled to solid state.
- The probability of moving to a higher energy state, instead of lower is: $p = e^{\frac{-\Delta E}{kT}}$

  where ΔE is the positive change in energy level, T is the temperature, and k is Boltzmann's constant.
- Temperature is high at the beginning.
- As temperature is lowered, probability of a downhill move gets smaller and smaller.
- If temperature is lowered very slowly, the best energy state is resulted.

Generic Concept of SA Algorithm

Procedure

Initialize Randomly generate a solution string

Evaluation (fitness function) for all solution string

Set 1. Initial and final temperature

2. Iterations at each temperature

While (Final temperature = Initial Temperature)

For (fixed number of iteration)

Randomly introduce a perturbation (a small change to the current solution string)

Evaluate newly generated string

Always accept the new alternative if it reduces the cost

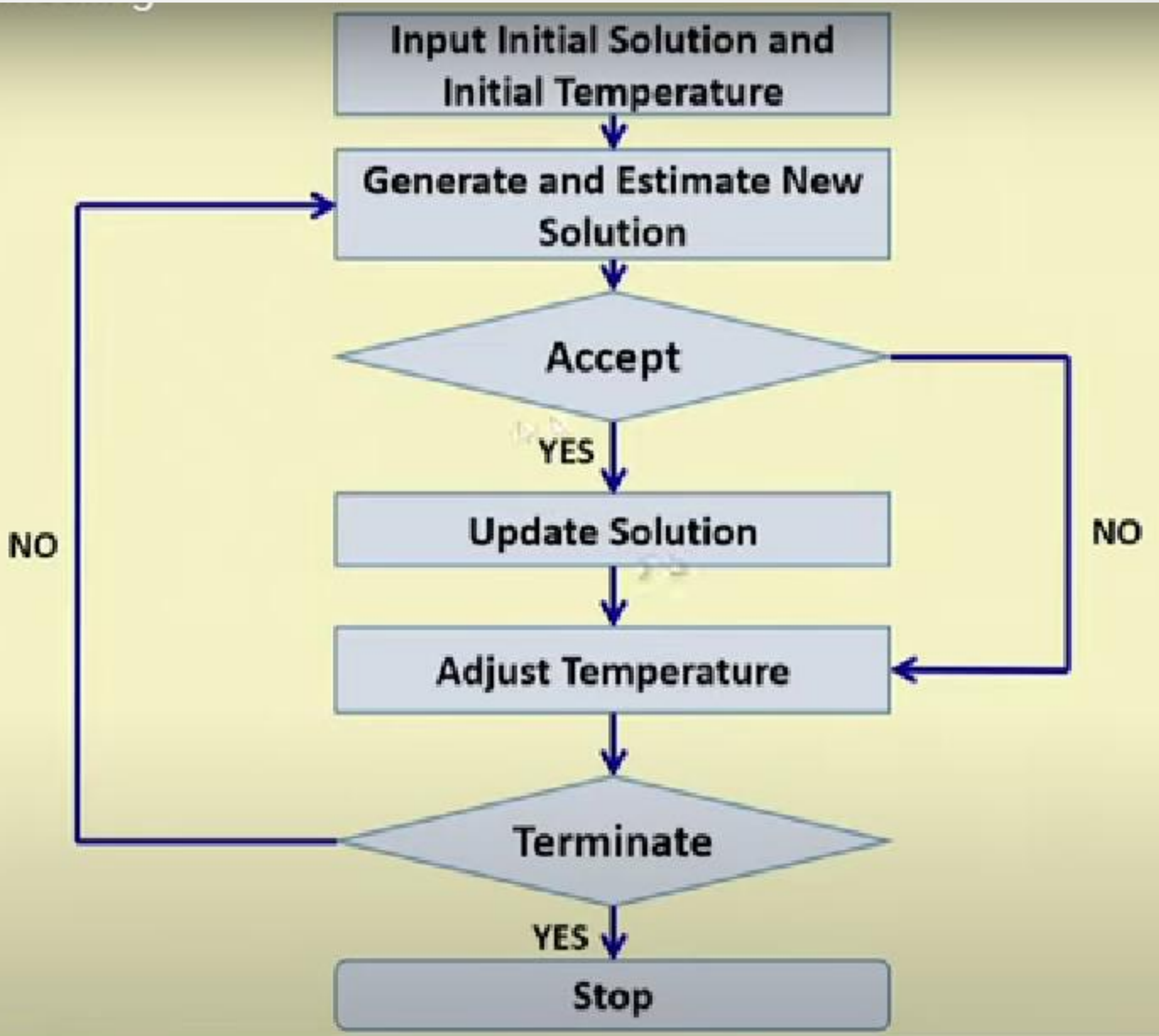
Randomly accept some alternatives that increase the cost

End of for loop

Reduction in final temperature

End

SA Steps


Input Initial Solution and Initial Temperature
Generate and Estimate New Solution
Accept
YES
NO
Update Solution
Adjust Temperature
NO
Terminate
YES
Stop

Simulated Annealing

Let **E** denotes the objective function value (also called **energy**)

If $\Delta E = E_{next} - E_{current} > 0$; probability of accepting a state with a **better object function** is always 1

If $\Delta E = E_{next} - E_{current} < 0$; probability of accepting a state with a **worse object function** is

$$P(\text{Accept Next}) = e^{\Delta E/T}$$

T = temperature at time step

For **maximizing**; slight change for **minimizing**

SA Steps

- Step-1: Choose an initial point $x^{(0)}$, a termination criterion ε. Set temperature T to a sufficiently high value and number of iterations n to be performed at a particular temperature.
- Step-2: Calculate a neighboring point $x^{(t+1)} = Nx^{(t)}$ randomly.
- Step-3 -If $\Delta E = E(x^{(t+1)}) - E(x^{(t)}) < 0$, set $t = t + 1$;
  Else create a random number r in the range (0,1).
  If $r \le \exp(-\Delta E / T)$, set $t = t + 1$; Else go to step 2.
- Step-4- If $\left|x^{(t+1)} - x^{(t)}\right| < \varepsilon$ and T is small, Terminate ;
  Else if (t mod n) = 0, Then lower T according to a coding schedule. Go to step 2

Simulated Annealing Example

Find the minimum value of the function using simulated annealing

$$Minimize\ \ f(x) = 500 - 20x_1 - 26x_2 - 4x_1x_2 + 4x_1^2 + 3x_2^2$$

$$-2 < x_1 < 10$$

$$-1 < x_2 < 11$$

Assume:

- **Temperature reduction factor c = 0.8**
- **Maximum permissible number of iteration as, n = 2.**

Simulated Annealing Example

- **Step 1: Calculate Initial Temperature T from 4 random points in the solution space:**

$$X^{(1)} = \begin{Bmatrix} 2 \\ 0 \end{Bmatrix}; \quad X^{(2)} = \begin{Bmatrix} 5 \\ 10 \end{Bmatrix}; \quad X^{(3)} = \begin{Bmatrix} 8 \\ 5 \end{Bmatrix}; \quad X^{(4)} = \begin{Bmatrix} 10 \\ 10 \end{Bmatrix};$$

$$f(x) = 500 - 20x_1 - 26x_2 - 4x_1x_2 + 4x_1^2 + 3x_2^2$$

$$f^{(1)} = 500 - 20(2) - 26(0) - 4(2)(0) + 4(2)^2 + 3(0)^2 = 476$$

$$Similarly, \quad f^{(2)} = 340; \quad f^{(3)} = 381; \quad f^{(4)} = 340$$

$$T = \frac{f^{(1)} + f^{(2)} + f^{(3)} + f^{(4)}}{4} = \frac{476 + 340 + 381 + 340}{4} = 384.25$$

Let the initial design point be: $X_1 = \begin{Bmatrix} 4 \\ 5 \end{Bmatrix}$

Simulated Annealing Example

Step 2: Evaluate the objective function value at $X_1 = \begin{Bmatrix} 4 \\ 5 \end{Bmatrix}$ as: $f_1 = f(X_1) = 349$

Set the iteration number as: $i = 1$

Step 3: Generate the new design point in the vicinity of the current design point.

Select two uniformly distributed random numbers: $u_1 = 0.31$ and $u_2 = 0.57$

with ±6 accuracy, variations are: x_1: (-2,10) and x_2: (-1,11)

so, $r_1 = -2 + u_1\{10 - (-2)\} = -2 + 0.31(12) = 1.72$

$r_2 = -1 + u_2\{11 - (-1)\} = -1 + 0.57(12) = 5.84$

New Design Point

$$X_2 = \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$$

As $f_2 = f(X_2) = 387.7312$; and $f_1 = 349$;

$\Delta f = f_2 - f_1 = 387.7312 - 349 = 38.7312$

Simulated Annealing Example

Step 4: As Δf is positive, Metropolis criterion is needed to accept or reject current point.

For this we choose a random number in the range (0,1) as r = 0.83.

Metropolis Criteria gives the probability of accepting the new design point X_2 as:

$$p[X_2] = e^{(-\Delta f / KT)}$$

By assuming the Boltzmann's constant K = 1, we have:

$$p[X_2] = e^{(-\Delta f/T)} = e^{(-38.7312/384.25)} = 0.9041$$

New Design Point

$$X_2 = \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$$

Since r = 0.83 is smaller than 0.9041, we accept the point X_2.

The objective function values f_2 is higher than f_1 in a minimization problem, X_2 is accepted as it is an early stage of simulation and current temperature is high.

Simulated Annealing Example

Step 2: Evaluate the objective function value at $X_2 = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$, as $f_2 = f(X_2) = 387.7312$

Set the iteration number as: $i = 2$

Step 3: Generate the new design point in the vicinity of the current design point.

Select two uniformly distributed random numbers: $u_1 = 0.92$ and $u_2 = 0.73$

with ±6 accuracy, we have: x_1: (-4.28,7.72) and x_2: (-0.16,11.84)

so, $r_1 = -4.28 + u_1\{7.72 - (-4.28)\} = -4.28 + 0.92(12) = 6.76$

$r_2 = -0.16 + u_2\{11.84 - (-0.16)\} = -0.16 + 0.73(12) = 8.60$

New Design Point

$$X_3 = \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 6.76 \\ 8.60 \end{Bmatrix}$$

As $f_3 = f(X_3) = 313.3264$; and $f_2 = 387.7312$;

$\Delta f = f_3 - f_2 = 313.3264 - 387.7312 = -74.4048$

Simulated Annealing Example

Step 4: As Δf is negative, Metropolis criterion is not needed.

we accept the current point X_3 and increase the iteration number to $i = 3$.

As $i > 2$, we go to step 5.

Step 5: Since a cycle of iteration with the current value of temperature is completed, we reduce the temperature to a new value:

New Temperature: $T_{new} = c * T_{old} = 0.5 * 384.25 = 192.125$

Reset the current iteration number as $i = 1$ and go to step 3.

Step 3: generate a new design point in the vicinity of the current design point X_3 and continue the procedure until the temperature reduced to a small value.

Simulated Annealing Drawbacks

- Although can avoid formation of any cycle, the rate of improvement is very low.
- SA does not have any memory to keep records of previously visited solution, hence their will always be a possibility for the search to return to such a solution again.