UNIVERSITY OF AGDER

ICT Seminar 2
Swarm Intelligence

Førsteamanuensis, Morten Goodwin, Ph.D.
2016-01-13

Overview

1. General course information
2. What is swarm intelligence?
   - Difficult problems
   - Examples from nature
3. Bird Flocking (Boids)
4. Particle Flying Model
5. Ant Colony Optimisation
   - Environment
   - Greedy
   - Standard ACO
   - ACO With Evaporation
   - MMAS
6. Multilevel ACO
   - Clustering
7. Classification using ACO— PolyACO

General course information

General Course

Grading — portfolio assessment:

- 25% Small assignments
- 75% Large project

Lecture Plan:

- `http://188.138.32.138/ikt441/LecturePlanIKT441.pdf`

What is swarm intelligence?

What is Swarm Intelligence

- "any attempt to design algorithms or distributed problem-solving devices inspired by the **collective behavior** of social insect colonies and other **animal societies**" [Bonabeau et al., 1999]

Source of image: `http://coolmoro2.tumblr.com`

What is swarm intelligence?—

Difficult problems

Traditional (machine learning) methods

- Problems need to be:
  - well-defined,
  - fairly predictable, and
  - computational within reasonable time

Traditional (machine learning) methods

- Problems need to be:
  - well-defined,
  - fairly predictable, and
  - computational within reasonable time
- Example: Classification of texts into topics.
  - Well-defined: What are the features? Words, N-GRAMS, ...
  - Fairly predictable: What is in the training data needs to be in the classification data as well.
  - Computational within reasonable time.

Why swarm intelligence?

- Many problems are fuzzy, not well defined.
- Many problems are unpredictable, non-deterministic.
- Many discrete optimization problems are NP hard.
- The problems are too large to solve completely.
  - Requires brute for for finding optimal solution.
  - Meta-heuristics, such as swarm intelligence, find good solutions.

NP

- Decision Problems in polynomial time on a non-deterministic touring machine.
- You have an efficient verifier (polynomial steps on a deterministic touring machine).
- Example: Subset-sum.

Small assignment

- Is there any nonempty subset of the following that sums to 0: $-3, -7, -2, 5, 8$

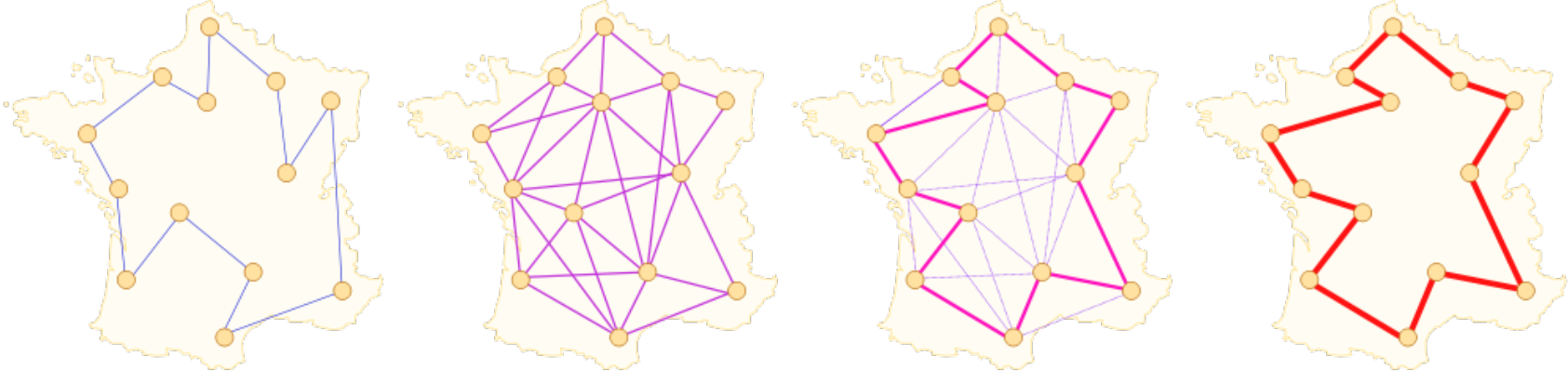
Small assignment

- Is there any nonempty subset of the following that sums to 0: $-3, -7, -2, 5, 8$
- Yes: $-3, -2, -2$

Small assignment

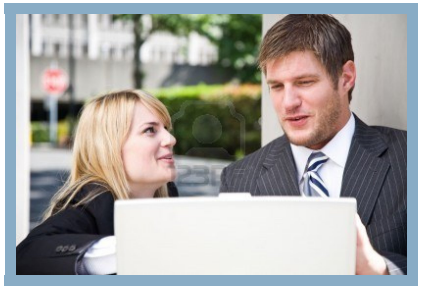
- Is there any nonempty subset of the following that sums to 0: $-3, -7, -2, 5, 8$
- Yes: $-3, -2, -2$
- Subset-sum problem.
- Efficient verifier: Summing numbers
- Yes answer: Find an example.
- No answer: Try every subset.

Traveling Salesmen Problem



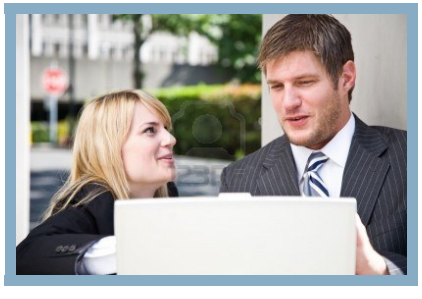
Source of image: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

P or NP?



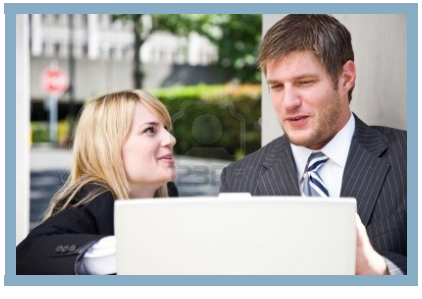
- NP-hard: Yes answer, find an example.
  No answer: Try every combination.
- Finding the longest path in a graph.
- How to best back N items of different weight into a knapsack with a limitation.
- Classifying a text into one 1 of 10 classes.

P or NP?



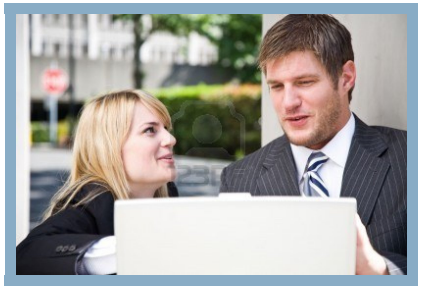
- NP-hard: Yes answer, find an example.
  No answer: Try every combination.
- Finding the longest path in a graph.
  - NP
- How to best back N items of different weight into a knapsack with a limitation.
- Classifying a text into one 1 of 10 classes.

P or NP?



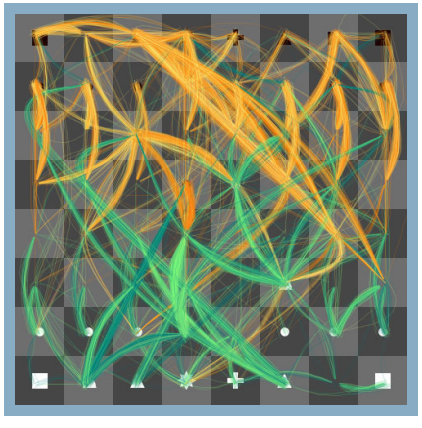
- NP-hard: Yes answer, find an example.
  No answer: Try every combination.
- Finding the longest path in a graph.
  - NP
- How to best back N items of different weight into a knapsack with a limitation.
  - NP
- Classifying a text into one 1 of 10 classes.

P or NP?



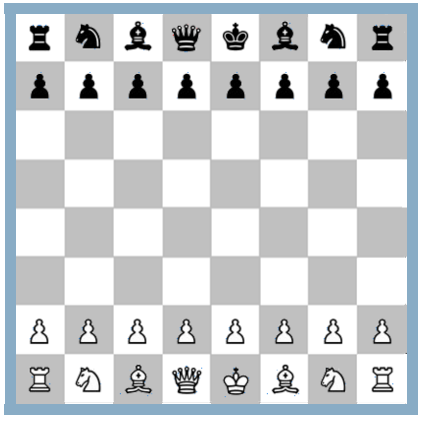
- NP-hard: Yes answer, find an example.
  No answer: Try every combination.
- Finding the longest path in a graph.
  - NP
- How to best back N items of different weight into a knapsack with a limitation.
  - NP
- Classifying a text into one 1 of 10 classes.
  - P

Other Hard Problems



Source of image:

http://ericmarcarelli.com/chess-ai-project/

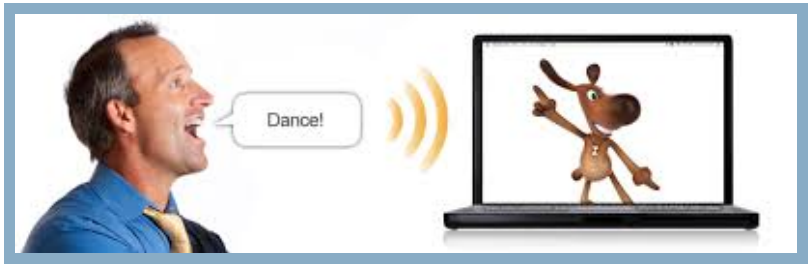


Source of image:

http://jimdavies.blogspot.no/2006/06/visualization-of-chess-ai.html

- Well-defined, but computationally difficult.
  - Travelling Salesman Problem — NP-hard.
  - Action-Response, .e.g. chess.

Other Hard Problems



Source of image:

`http://freesourcecode.net/matlabprojects/58096/speech-recognition-matlab-code`

- Fuzzy problems
  - Speech detection

Other Hard Problems

Source of image:

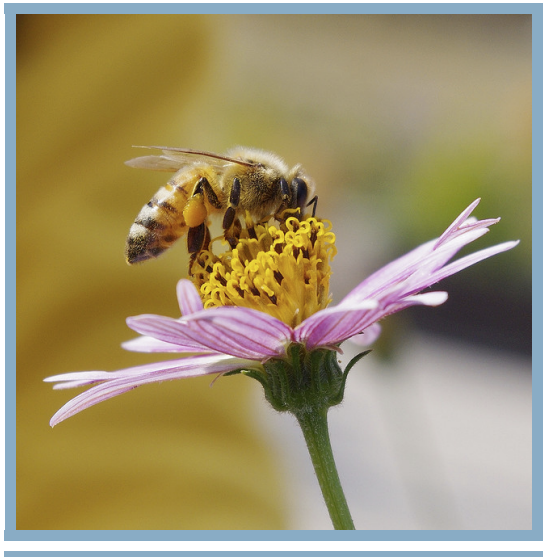
https://iqoption.com/promo/binary-options_en/?aff=20086&afftrack=mixed_eu_

- Hardly predicable problems
  - Stock exchange prediction
  - Autonomous robots

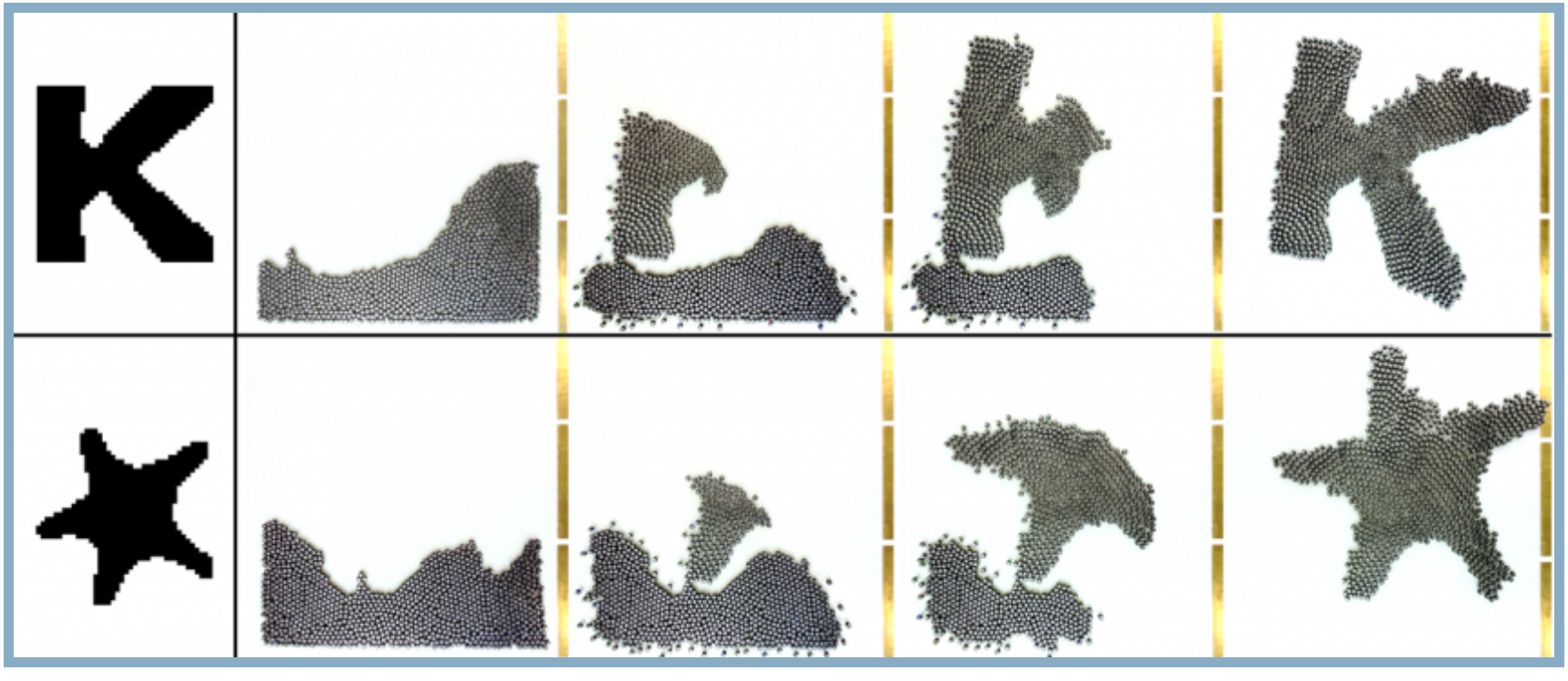
What is swarm intelligence?—

Examples from nature

Swarm Intelligence — Examples from nature



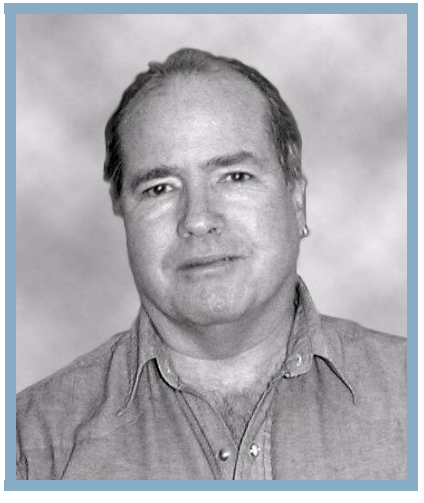
Robot Swarm Intelligence



http://phys.org/news/2014-08-autonomous-robots-self-organizing-thousand-robot-s.html

Inventors — Swarm Intelligence

- Russel Eberhart
- Electrical Engineer



- James Kennedy
- Social Psychologist

What is Swarm Intelligence?

- Robust Stochastic Optimization Technique.
- A Meta-heuristic technique.
- Based on simple movements of individuals yielding intelligent swarms.
- Rather informamal, proofs based on
- **Robost:** Does not depend on single point of failures.
- **Distributed decision:** Collective decision
- **Emergent:** Solving fairly complex task.

Meta-heuristics

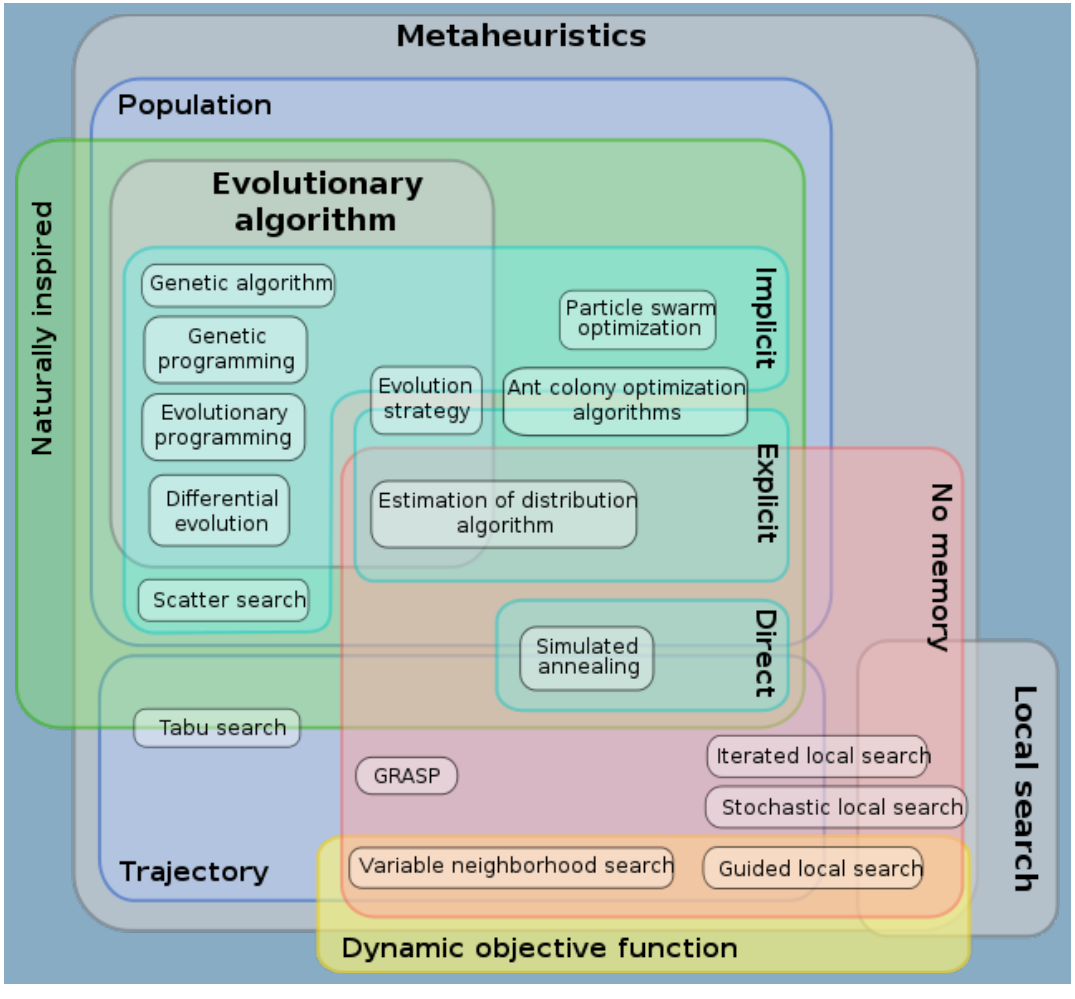
- No guarantee for finding optimal solution.
- Other Meta-heuristics
  - Genetic Algorithms
  - Variable neighbour search.
  - Evolutionary computing.
  - Simulated annealing.
- A technique to find an heuristic.

Heuristic vs Meta-heuristic

- **Heuristic**
- Often Problem dependent
- Often greedy, or very close to greedy
- Often gets trapped in local minima.
- Example: A*

- **Meta-heuristic**
- Often Problem independent
- Often not greedy
- Seldom gets trapped in local minima.
- Example: ACO

Meta-heuristics



Source of image: `https://en.wikipedia.org/wiki/Metaheuristic`

Bird Flocking (Boids)

Flocking in movies

https://www.youtube.com/watch?v=9v_UCB_qwPc

Basics I

```
1  import random
2  import math
3
4  class Boid:
5      def __init__(self):
6          self.x = random.uniform(0,100)
7          self.y = random.uniform(0,100)
8          self.speedx = 1.0
9          self.speedy = 1.0
10
11     def move(self):
12         self.x += self.speedx
13         self.y += self.speedy
14
15
```

Basics II

```
16  allboids = [Boid() for i in range(100)]
17  def getNextBoids():
18      for boid in allboids:
19          boid.move()
20      return [b.x for b in allboids],[b.y for b in allboids]
```

Adding a goal — speed I

```
1  class Boid:
2      def __init__(self):
3          self.x = random.uniform(0,100)
4          self.y = random.uniform(0,100)
5          self.speedx = 1.0
6          self.speedy = 1.0
7          self.lastx = self.x-1
8          self.lasty = self.y-1
9
10     def move(self):
11         self.lastx = self.x
12         self.lasty = self.y
13         self.x += self.speedx
14         self.y += self.speedy
15
```

Adding a goal — speed II

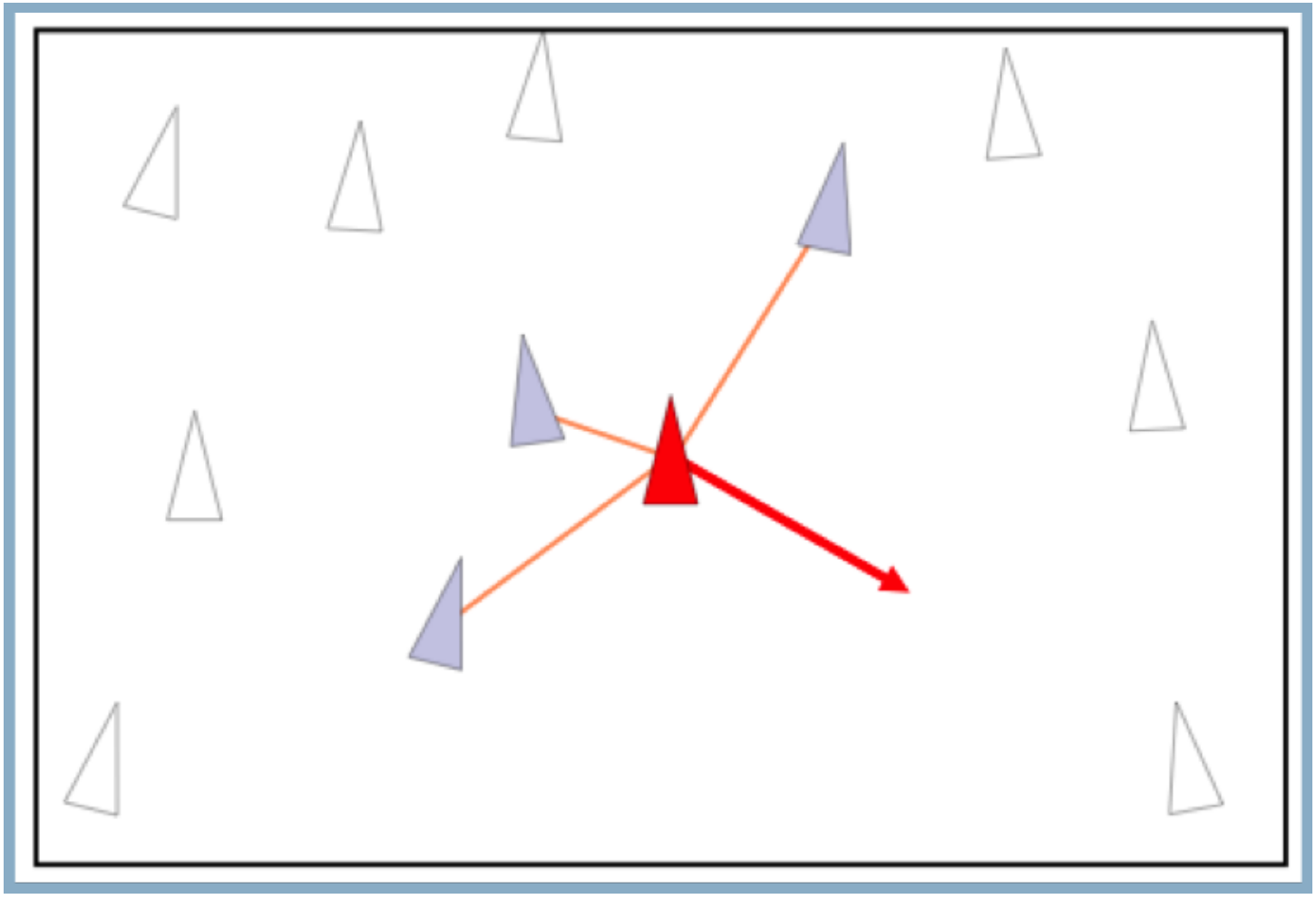
```
16      def calcSpeed(self):
17          hypotenuse = abs(math.sqrt((self.x-self.lastx)**2 + (self.y-self.lasty)**2))
18          if(hypotenuse<0.5):
19              hypotenus = 0.5
20          if(hypotenuse>2):
21              hypotenuse = 2
22          catheti = math.sqrt((hypotenuse**2)/2)
23
24          if(self.x>80):
25              self.speedx = -catheti
26          if(self.x<20):
27              self.speedx = catheti
28          if(self.y>80):
29              self.speedy = -catheti
30          if(self.y<20):
```

Adding a goal — speed III

```
31              self.speedy = catheti
32
33
34
35
36  allboids = [Boid() for i in range(100)]
37  def getNextBoids():
38      for boid in allboids:
39          boid.calcSpeed()
40          boid.move()
41      return [(b.x,b.y) for b in allboids]
```

Boid - Rule 1

- Avoid collision with neighbours



Avoid crash I

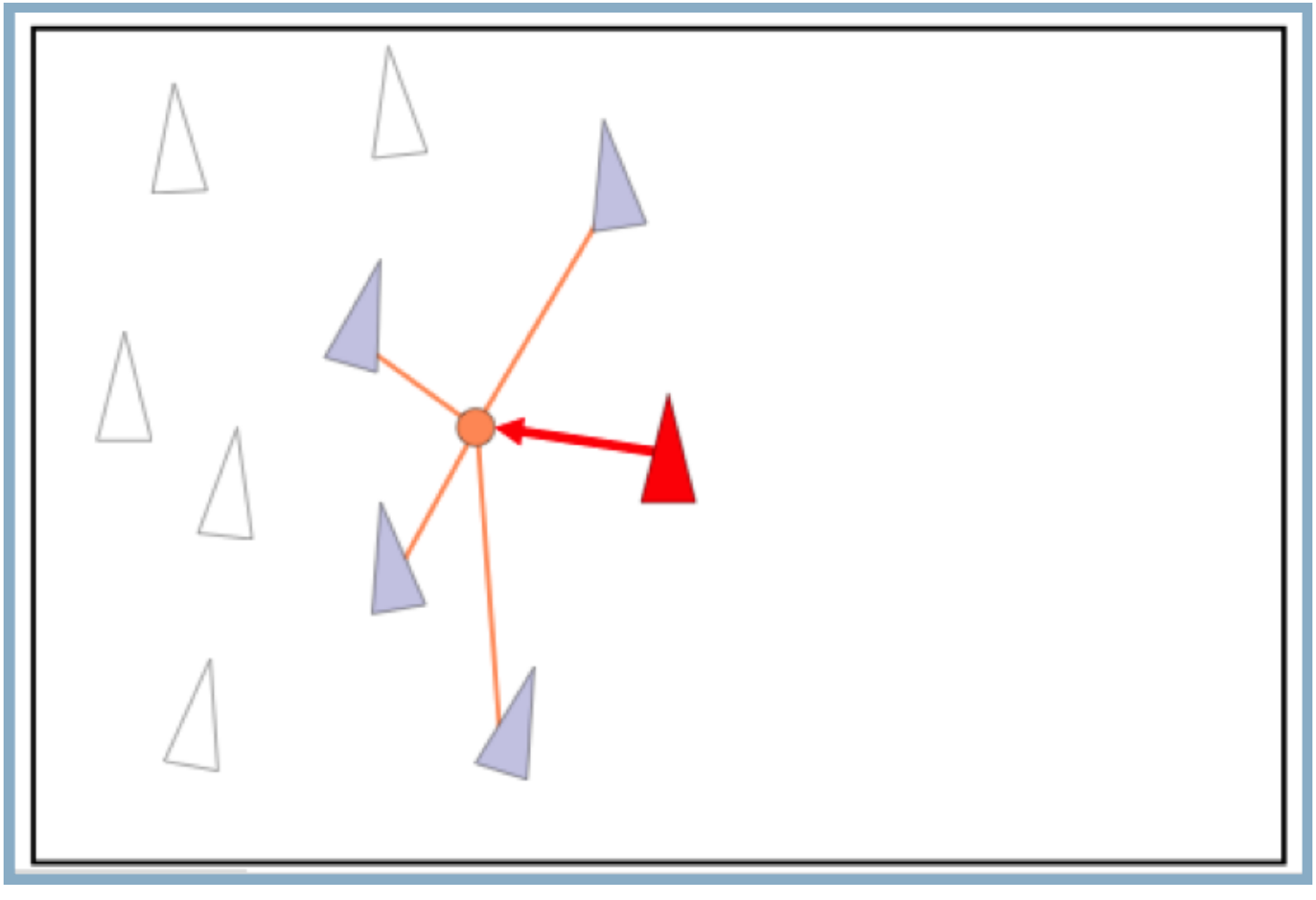
```
1   def move(self):
2       self.lastx = self.x
3       self.lasty = self.y
4       self.x += self.speedx
5       if(self.isCrashing()):
6           self.x -= self.speedx
7
8       self.y += self.speedy
9       if(self.isCrashing()):
10          self.y -= self.speedy
11
12  def isCrashing(self):
13      for oneBoid in allboids:
14          if(oneBoid != self):
15              crash = True
```

Avoid crash II

```
16              if(self.x>oneBoid.x+2 or oneBoid.x>self.x+2):
17                  crash = False
18              if(self.y>oneBoid.y+2 or oneBoid.y>self.y+2):
19                  crash = False
20              if(crash):
21                  return True
22          return False
23
24      def avoidCrashOthers(self):
25          while(self.isCrashing()):
26              self.x += self.speedx
27              self.y += self.speedy
```

Boid - Rule 2

- Stay near neighbours



Stay close to neighbours I

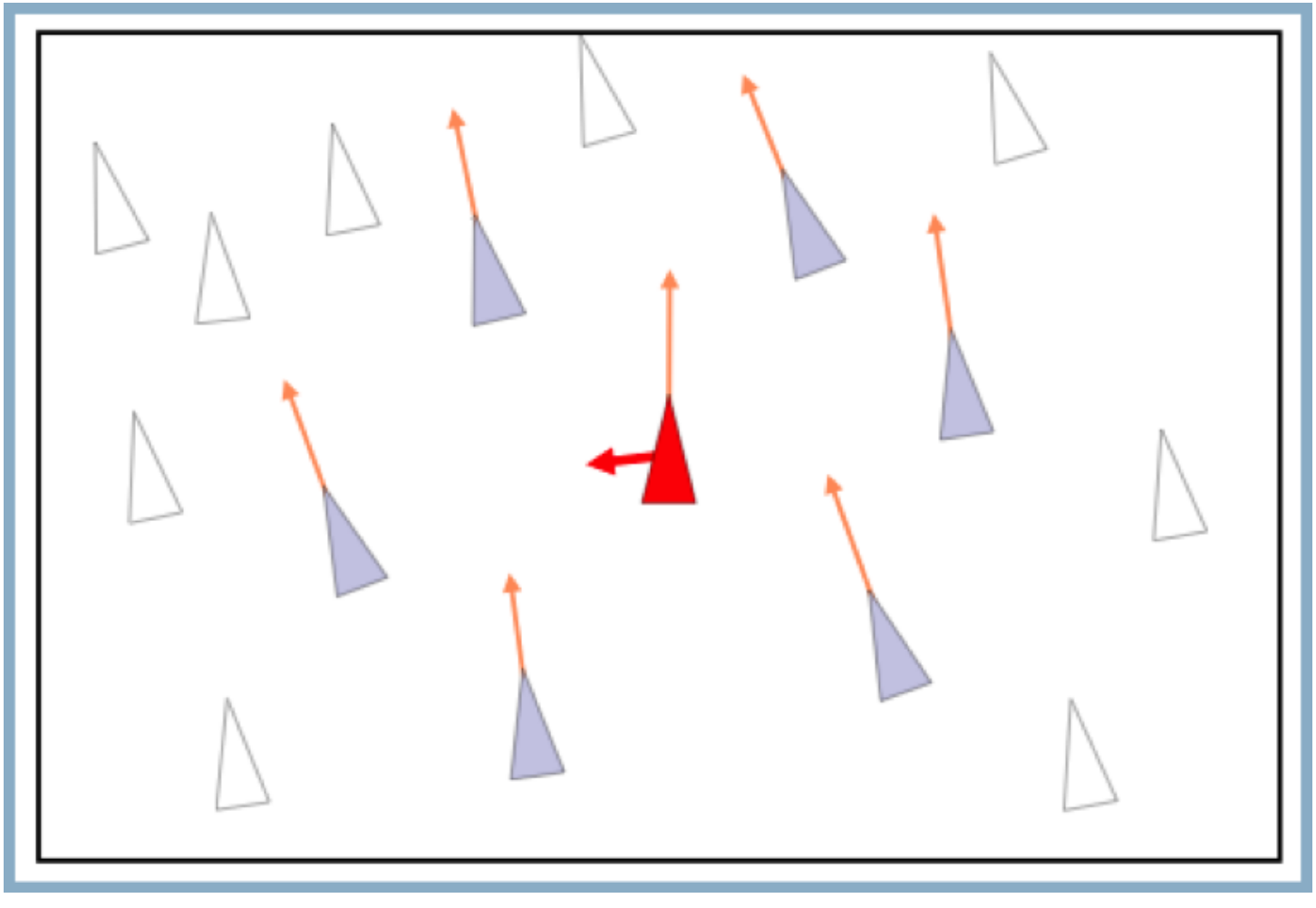
```
1   def getDistance(self,other):
2       return abs(math.sqrt((self.x−other.x)**2 + (self.y−other.y)**2))
3
4   def getClosest(self):
5       closestBoid = allboids[0]
6       distance = self.getDistance(closestBoid)
7       for oneBoid in allboids:
8           if oneBoid != self:
9               if(self.getDistance(oneBoid)<distance):
10                  closestBoid = oneBoid
11                  distance = self.getDistance(closestBoid)
12      return closestBoid,distance
13
14  def stayCloseToOthers(self):
15      closestBoid,distance = self.getClosest()
```

Stay close to neighbours II

```
16          if(distance<5 and distance>2):
17              if(closestBoid.x<self.x):
18                  self.x -= 1
19              if(closestBoid.y<self.y):
20                  self.y -= 1
21              if(closestBoid.x>self.x):
22                  self.x += 1
23              if(closestBoid.y>self.y):
24                  self.y += 1
```

Boid - Rule 3

- Match the velocity of neighbours



Match Speed

```
1    def mapClosestSpeed(self):
2        closestBoid,distance = self.getClosest()
3        if(distance<5):
4            self.speedx = closestBoid.speedx
5            self.speedy = closestBoid.speedy
```

Complete Boids I

```
1  import random
2  import math
3
4  class Boid:
5      def __init__(self):
6          self.y = random.uniform(0,100)
7          self.x = random.uniform(0,100)
8          self.movementx = 1.0
9          self.movementy = 1.0
10         self.lastx = self.x-1
11         self.lasty = self.y-1
12
13     def getDistance(self,other):
14         return abs(math.sqrt( (self.x - other.x)**2 + (self.y - other.y)**2 ))
15
```

Complete Boids II

```
16
17      def calcSpeed(self):
18          self.speed = abs(math.sqrt( (self.x - self.lastx)**2 + (self.y - self.lasty)**2 ))
19          if self.speed< 0.5:
20              self.speed = 0.5
21          if self.speed> 2:
22              self.speed = 2
23
24          #print "Speed:",self.speed
25          if(self.x>80):
26              self.movementx = -math.sqrt((self.speed**2)/2)
27          if(self.x<20):
28              self.movementx = math.sqrt((self.speed**2)/2)
29          if(self.y>80):
30              self.movementy = -math.sqrt((self.speed**2)/2)
```

Complete Boids III

```
31          if(self.y<20):
32              self.movementy = math.sqrt((self.speed**2)/2)
33          #self.lastx = self.x
34          #self.lasty = self.y
35
36
37      def move(self):
38          self.lastx = self.x
39          self.lasty = self.y
40          self.x += self.movementx
41          self.y += self.movementy
42          if(self.crashOthers()):
43              self.x -= self.movementx
44              self.y -= self.movementy
45          #self.calcSpeed()
46
```

Complete Boids IV

```
47
48      def avoidCrashOthers(self):
49          i = 0
50          while(self.crashOthers() and i<5):
51              i+=1
52              #self.movementy = - self.movementy
53              #self.movementx = - self.movementx
54              #print "Crashing"
55              self.x+=self.movementx#random.sample([-self.movementx,self.↩
            movementx],1)[0]
56              self.y+=self.movementy#random.sample([-self.movementy,self.↩
            movementy],1)[0]
57              #self.movementx *=2
58              #self.movementy *=2
59      def avoidOutsideBoard(self):
60          if(self.x<0):
```

Complete Boids V

```
61              #self.x = 100
62              self.movementx = abs(self.movementx)
63              self.x+=1
64          if(self.x>100):
65              #self.x = 0
66              self.movementx = -abs(self.movementx)
67              self.x-=1
68          if(self.y<0):
69              #self.y = 100
70              self.movementy = abs(self.movementy)
71              self.y +=1
72          if(self.y>100):
73              #self.y = 0
74              self.movementy = -abs(self.movementy)
75              ##self.y -=1
76
```

Complete Boids VI

```
77      def mapSpeedToClosest(self):
78          closestBoid = allboids[0]
79          distance = self.getDistance(closestBoid)
80          for oneBoid in allboids:
81              if oneBoid != self:
82                  if(self.getDistance(oneBoid)<distance):
83                      closestBoid = oneBoid
84                      distance = self.getDistance(oneBoid)
85          if(distance<10):
86              self.movementx = closestBoid.movementx
87              self.movementy = closestBoid.movementy
88
89
90      def stayCloseToOthers(self):
91          closestBoid = allboids[0]
92          distance = self.getDistance(closestBoid)
```

Complete Boids VII

```
93          for oneBoid in allboids:
94              if oneBoid != self:
95                  if(self.getDistance(oneBoid)<distance):
96                      closestBoid = oneBoid
97                      distance = self.getDistance(oneBoid)
98          if(distance<5):
99              if(closestBoid.x<self.x):
100                 self.x -= 1
101             if(closestBoid.x>self.x):
102                 self.x += 1
103             if(closestBoid.y<self.y):
104                 self.y -= 1
105             if(closestBoid.y>self.y):
106                 self.y += 1
107
108     def crashOthers(self):
```

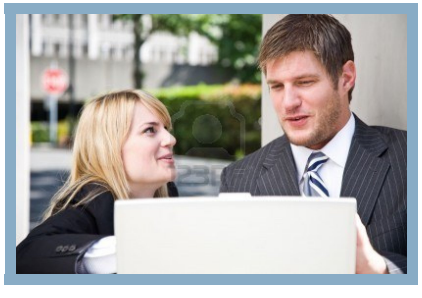
Complete Boids VIII

```
109             for oneBoid in allboids:
110                 if oneBoid != self:
111                     crash = True
112                     if(self.x>oneBoid.x+2 or oneBoid.x>self.x+2):
113                         crash = False
114                     if(self.y>oneBoid.y+2 or oneBoid.y>self.y+2):
115                         crash = False
116                     if crash:
117                         return True
118
119
120 allboids = [Boid() for i in range(100)]
121
122
123 def getNextBoids():
124     #random.shuffle(allboids)
```

Complete Boids IX

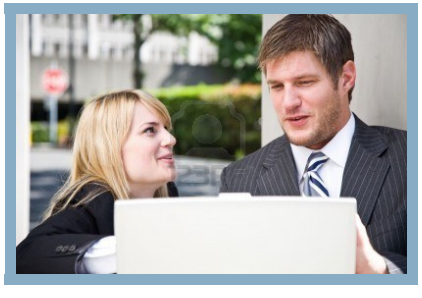
```
125     for boid in allboids:
126         #boid.mapSpeedToClosest()
127         boid.calcSpeed()
128         boid.move()
129         boid.avoidCrashOthers()
130         boid.stayCloseToOthers()
131         #boid.avoidOutsideBoard()
132     return [(b.x,b.y) for b in allboids]
```

Short discussion



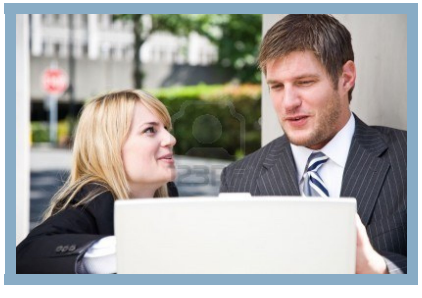
- Is it robust?

Short discussion



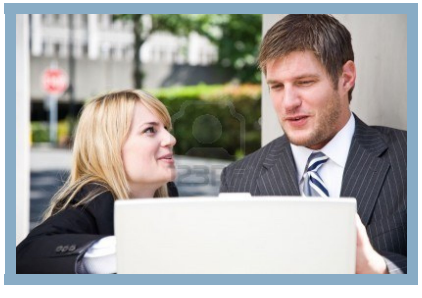
- Is it robust?
- Yes, no central control.

Short discussion



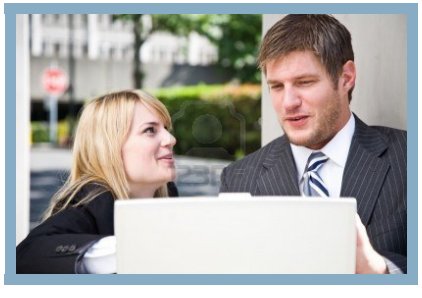
- Is it emergent?

Short discussion



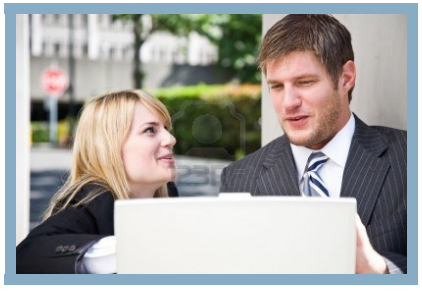
- Is it emergent?
- Yes, can perform fairly complex tasks without specifically programming for it.

Short discussion



- Does it have a distributed decision?

Short discussion



- Does it have a distributed decision?
- Yes, at least no central decicision

Characteristics

- Simple and equal rules to all individuals.
- No central control.
  - Robust.
- Emergent.
  - Can do fairly complex tasks.

Particle Flying Model

Particle Flying Model

Source of image:

http://bit.ly/1IQtAlq

- Random agents creating swarms around agents.
- Particles are points in N-dimensional space.

Particle Flying Model — Pseudo code

```
1 While not stop criteria met
2     For each particle
3         Calculate fitness
4         If fitness is better than the best fitness (pbest), set the current fitness↩
          tp pbest.
5     Choose the particle with best fitness at gbest
6     For each particle
7         Calculate particle velocity $v_i^{k+1}$.
8         Calculate particle position $s_i^{k+1}$.
```

Environment

```
1  import random
2  import math
3  class Environment:
4      def __init__(self,x,y):
5          self.x = x
6          self.y = y
7
8      def distance(self,x,y):
9           return math.sqrt((x-self.x)**2 + (y-self.y)**2)
10 e = Environment(10,10)
```

Particle

```
1  class Particle:
2      def __init__(self):
3          self.x,self.y = random.uniform(0,100),random.uniform(0,100)
4
5  particles = [Particle() for i in range(1000)]
6
7  def getNextParticles():
8      return [p.x for p in particles],[p.y for p in particles]
```

Particle Flying Model

Source of image:

http://bit.ly/1IQtAlq

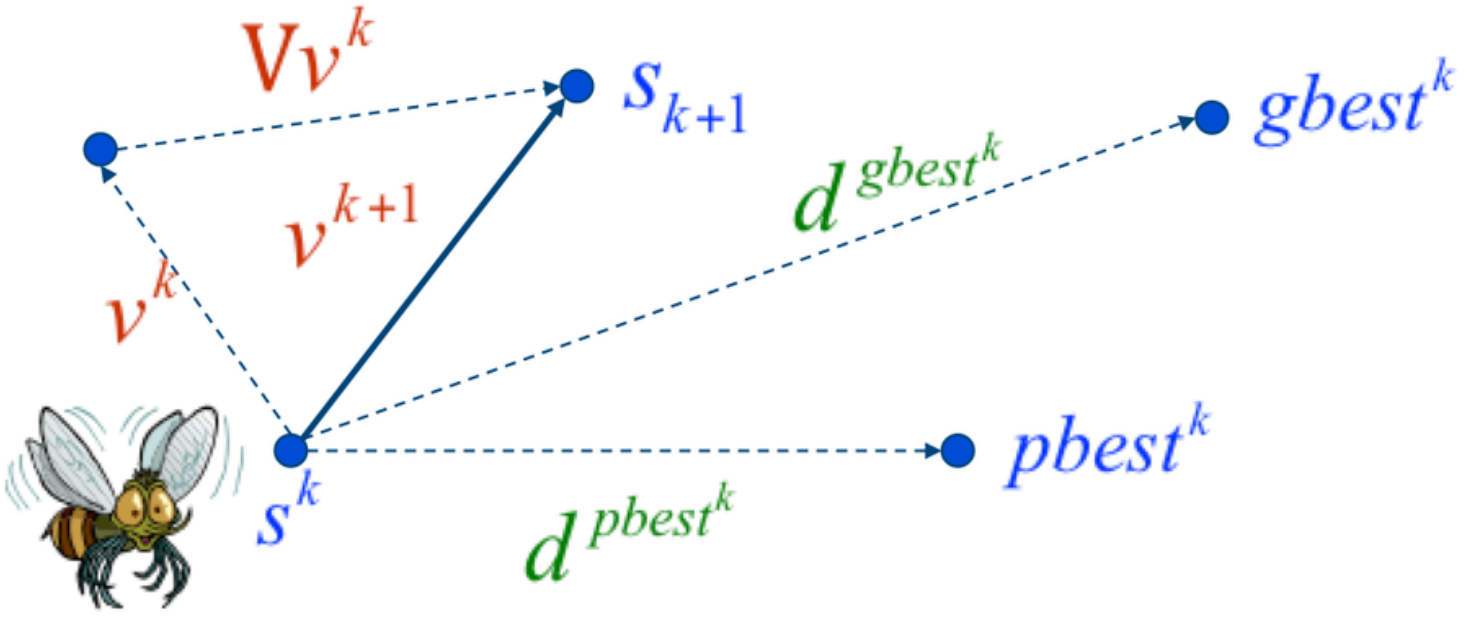
- p_{best}: The best solution by that particle.
- g_{best}: The best solution by all particles.
- The particles should be accelerated towards p_{best} and g_{best}.

Particle Flying Model

$$Vv^k = w_1 d^{pbest^k} + w_2 d^{gbest^k}$$

$$w_1 = c_1 \cdot rand()$$

$$w_2 = c_2 \cdot rand()$$



[Yue, 2015]

Pbest

```
1  class Particle:
2      def __init__(self):
3          self.x,self.y = random.uniform(0,100),random.uniform(0,100)
4          self.pbestx,pbesty = self.x,self.y
5          self.pbest = 100*100
6
7      def updateBest(self):
8          d = e.distance(self.x,self.y)
9          if(d<self.pbest):
10             self.pbest = d
11             self.pbestx,self.pbesty = self.x,self.y
```

Gbest I

```
1  e = Environment(10,10)
2  global gbest,gbestx,gbesty
3  gbest = 100*100
4  gbestx,gbesty = random.uniform(0,100),random.uniform(0,100)
5  class Particle:
6      def __init__(self):
7          self.x,self.y = random.uniform(0,100),random.uniform(0,100)
8          self.pbestx,pbesty = self.x,self.y
9          self.pbest = 100*100
10
11     def updateBest(self):
12         d = e.distance(self.x,self.y)
13         if(d<self.pbest):
14             self.pbest = d
15             self.pbestx,self.pbesty = self.x,self.y
```

Gbest II

```
16          global gbest,gbestx,gbesty
17          if(d<gbest):
18              gbest = d
19              gbestx,gbesty = self.x,self.y
```

Particle Flying Model

$$Vv^k = w_1 d^{p^k_{best}} + w_2 d^{g^k_{best}} \tag{1}$$

where

$$w_1 = c_1 * rand() \tag{2}$$

$$w_2 = c_2 * rand() \tag{3}$$

$$\delta v^k = c_1 * \textit{rand()} * (p^k_{besti} - s^k_i) + c_2 * \textit{rand()} * (g^k_{besti} - s^k_i) \tag{4}$$

Calculate velocity

```
1  def calcVelocity(self):
2      #Delta V
3      w1,w2 = C1*random.uniform(-1,1),C2*random.uniform(-1,1)
4      deltav_x = w1*(self.pbestx-self.x) + w2*(gbestx-self.x)
5      deltav_y = w1*(self.pbesty-self.y) + w2*(gbesty-self.y)
6
7      #V
8      self.v_x = deltav_x
9      self.v_y = deltav_y
```

Particle Flying Model

For particle i

$$s_i^{k+1} = s_i^k + v_i^{k+1} \tag{5}$$

$$v_i^{k+1} = v_i^k + \delta v_i^k \tag{6}$$

$$Vv^k = w_1 d^{p_{best}^k} + w_2 d^{g_{best}^k} \delta v^k = c_1 * rand() * (p_{besti}^k - s_i^k) + c_2 * rand() * (g_{besti}^k - \tag{7}$$

Moving

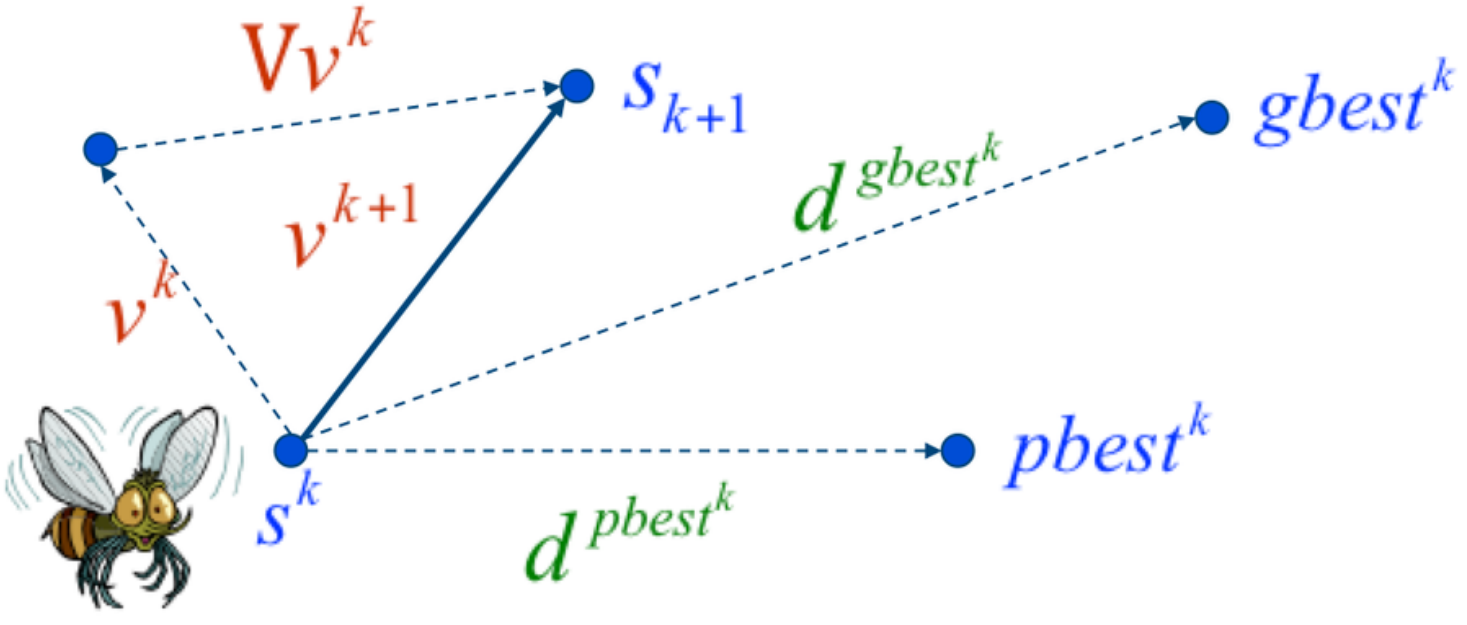
```
1    def move(self):
2        self.x += self.v_x
3        self.y += self.v_y
```

Particle Flying Model

$$Vv^k = w_1 d^{pbest^k} + w_2 d^{gbest^k}$$

$$w_1 = c_1 \cdot rand()$$

$$w_2 = c_2 \cdot rand()$$



[Yue, 2015]

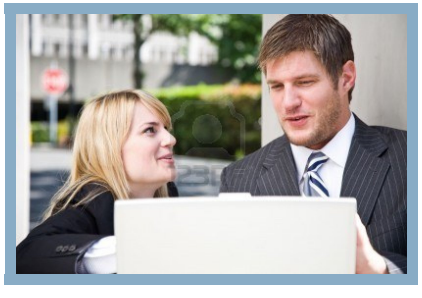
Particle Flying Model

- Each particle tries to modify its position based on:
  - Its current possition, s^k
  - Its current velocity, v^k
  - The distance between s^k and p_{best}
  - The distance between s^k and g_{best}

Stochastic

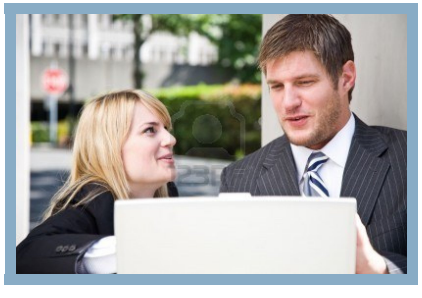
```
1  det = raw_input("Deterministic")=="y"
2
3  class Environment:
4      def __init__(self,x,y):
5          self.x = x
6          self.y = y
7
8      def distance(self,x,y):
9          thisx = random.uniform(-1,1)+self.x
10         thisy = random.uniform(-1,1)+self.y
11         if det:
12             this.x = self.x
13             this.y = self.y
14         return math.sqrt((x-thisx)**2 + (y-thisy)**2)
```

Short discussion



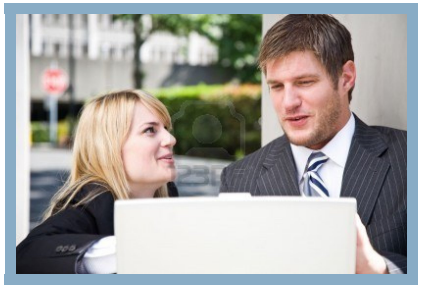
- Is it robust?

Short discussion



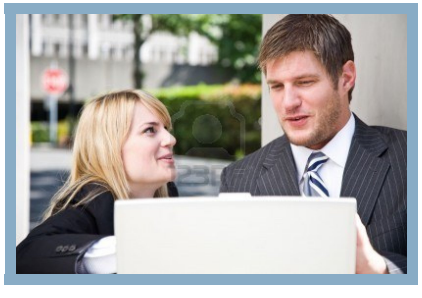
- Is it robust?
- Yes, no central control. Does not converge to a local optima.

Short discussion



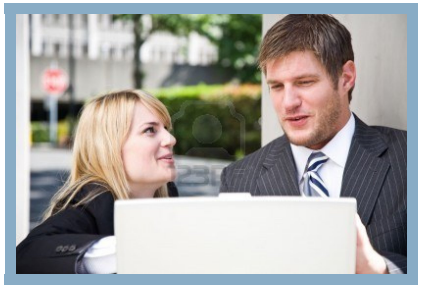
- Is it emergent?

Short discussion



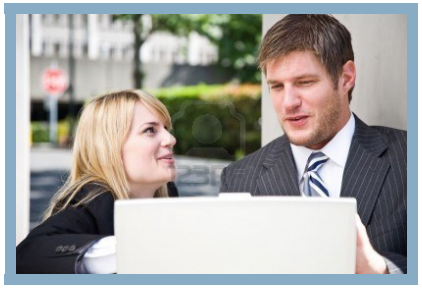
- Is it emergent?
- Yes, can perform fairly complex tasks without specifically programming for it.

Short discussion



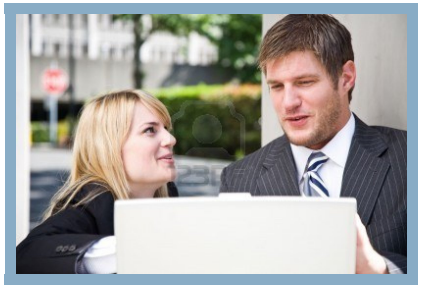
- Does it have a distributed decision?

Short discussion



- Does it have a distributed decision?
- Yes, at least no central decicision

Short discussion



- Practical examples where particle swarm optimisation may be useful?

Example of usages

- Parameter esitmation.
  - Location of solar panel.
  - Other ways of doing parameter estimation?

Example of usages

- Parameter esitmation.
  - Location of solar panel.
  - Other ways of doing parameter estimation?
  - Sampling.

Ant Colony Optimisation

Ant Colony Optimization

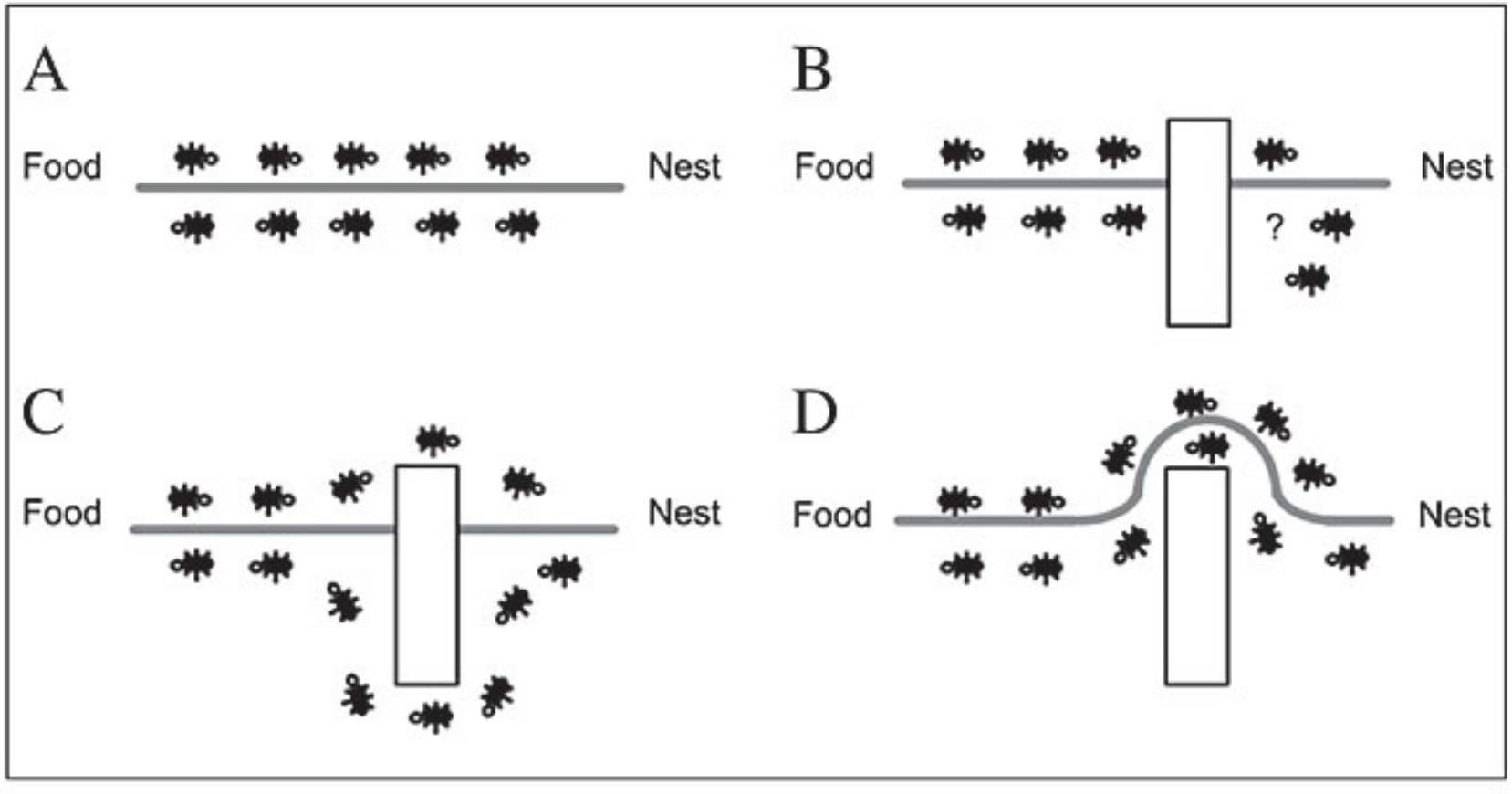
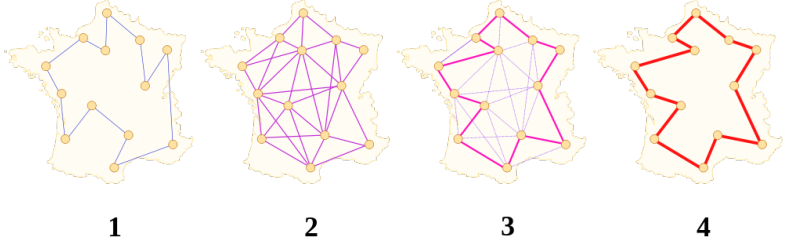


Figure 2. A. Ants in a pheromone trail between nest and food; **B.** an obstacle interrupts the trail; **C.** ants find two paths to go around the obstacle; **D.** a new pheromone trail is formed along the shorter path.

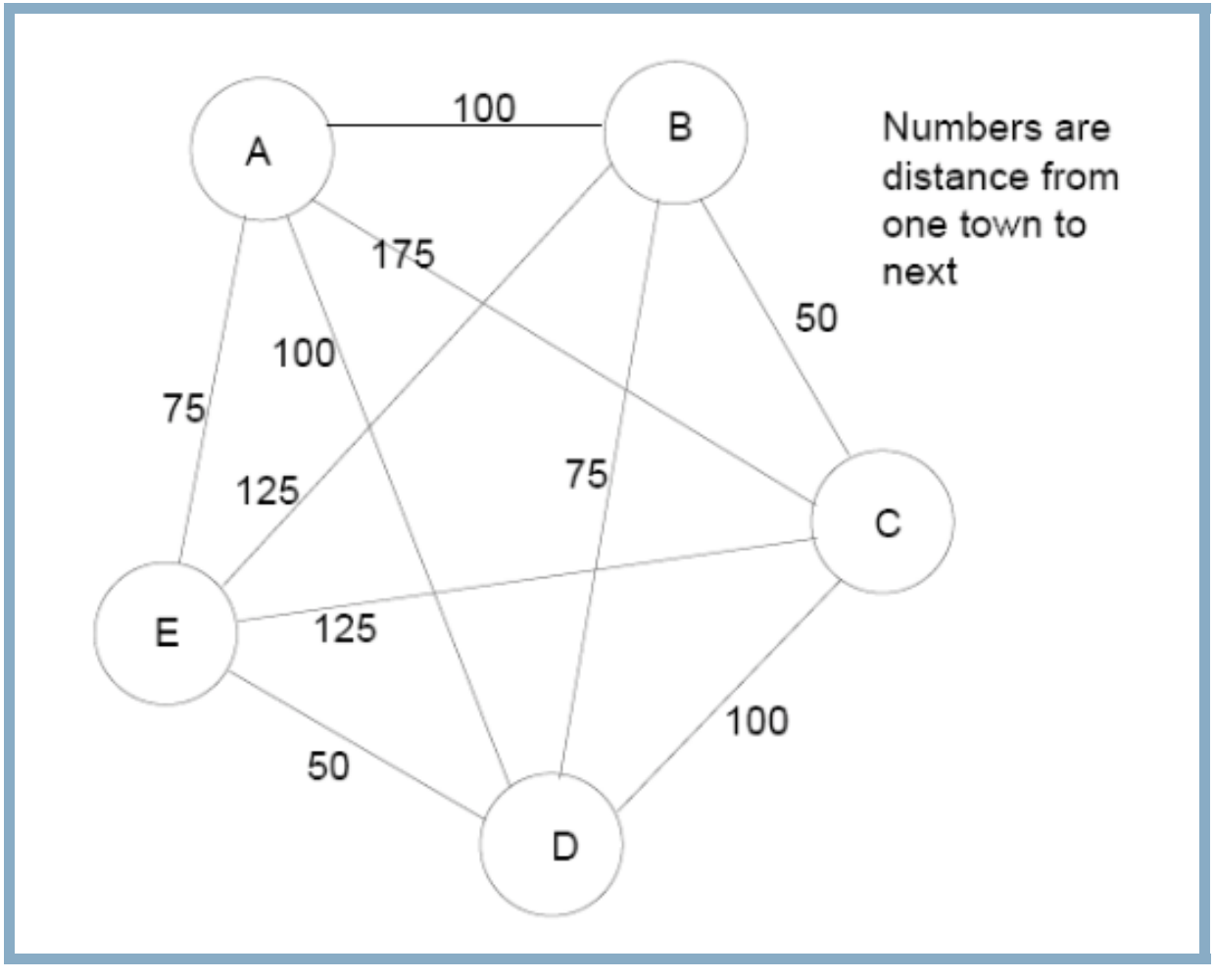
ACO Traveling Salesman

1. Ant chooses one path.
2. Ants lay pheromone trails.
3. Pheromones reinforce good path.
4. Best path emerges.



Source of image: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

Problem



- Find a tour that minimizes the traveling to all nodes

Source of image: http://www.inf.ed.ac.uk/teaching/courses/nat/slides/lecture10_ACO.pdf

Plans

- ACO.
- ACO with evaporation
- ACO MMAS

Ant Colony Optimisation—
Environment

Environment I

```
1  import random
2  class Node:
3      def __init__(self,name):
4          self.name = name
5          self.edges = []
6
7      def rouletteWheelSimple(self):
8          return random.sample(self.edges,1)[0]
9
10     def __repr__(self):
11         return self.name
12
13 class Edge:
14     def __init__(self,fromNode,toNode,cost):
15         self.fromNode = fromNode
```

Environment II

```
16          self.toNode = toNode
17          self.cost = cost
18
19      def __repr__(self):
20          return self.fromNode.name + "--(" + str(self.cost) + ")--" + self.↩
            toNode.name
21
22  a = Node("A")
23  b = Node("B")
24  c = Node("C")
25  d = Node("D")
26  e = Node("E")
27
28  nodes = [a,b,c,d,e]
29  edges = [
30      Edge(a,b,100),
```

Environment III

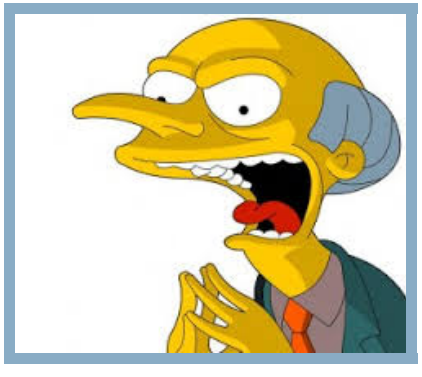
```
31      Edge(a,c,175),
32      Edge(a,d,100),
33      Edge(a,e,75),
34      Edge(b,c,50),
35      Edge(b,d,75),
36      Edge(b,e,125),
37      Edge(c,d,100),
38      Edge(c,e,125),
39      Edge(d,e,75)]
40
41  #Make symetrical
42  for oneEdge in edges[:]:
43      edges.append(Edge(oneEdge.toNode,oneEdge.fromNode,oneEdge.cost))
44
45
46  #Assign to nodes
```

Environment IV

```
47  for oneEdge in edges:
48      for oneNode in nodes:
49          if(oneEdge.fromNode==oneNode):
50              oneNode.edges.append(oneEdge)
```

Ant Colony Optimisation—
Greedy

Greedy I



Source of image: `http://shellaeversey.com`

1. For every Node:
   - Select the edge with the lowest cost leading to a Node not previously visited.

Greedy I

```
1  def checkAllNodesPresent(edges):
2      visitedNodes = [edge.toNode for edge in edges]
3      return set(nodes).issubset(visitedNodes)
4
5  class Greedy:
6      def __init__(self):
7          self.visitedEdges = []
8          self.visitedNodes = []
9
10     def walk(self,startNode):
11             currentNode = startNode
12             currentEdge = None
13             while(not checkAllNodesPresent(self.visitedEdges)):
14                 possibleEdges = [(edge.cost,edge) for edge in currentNode.edges ↩
           if edge.toNode not in self.visitedNodes]
```

Greedy II

```
15              possibleEdges.sort()
16              #import pdb;pdb.set_trace()
17              currentEdge = possibleEdges[0][1]
18              currentNode = currentEdge.toNode
19              self.visitedEdges.append(currentEdge)
20              self.visitedNodes.append(currentNode)
21              print currentNode,currentEdge
22
23  g = Greedy()
24  g.walk(a)
25  print "Cost:",sum([e.cost for e in g.visitedEdges])
```

Hand-raising assignment

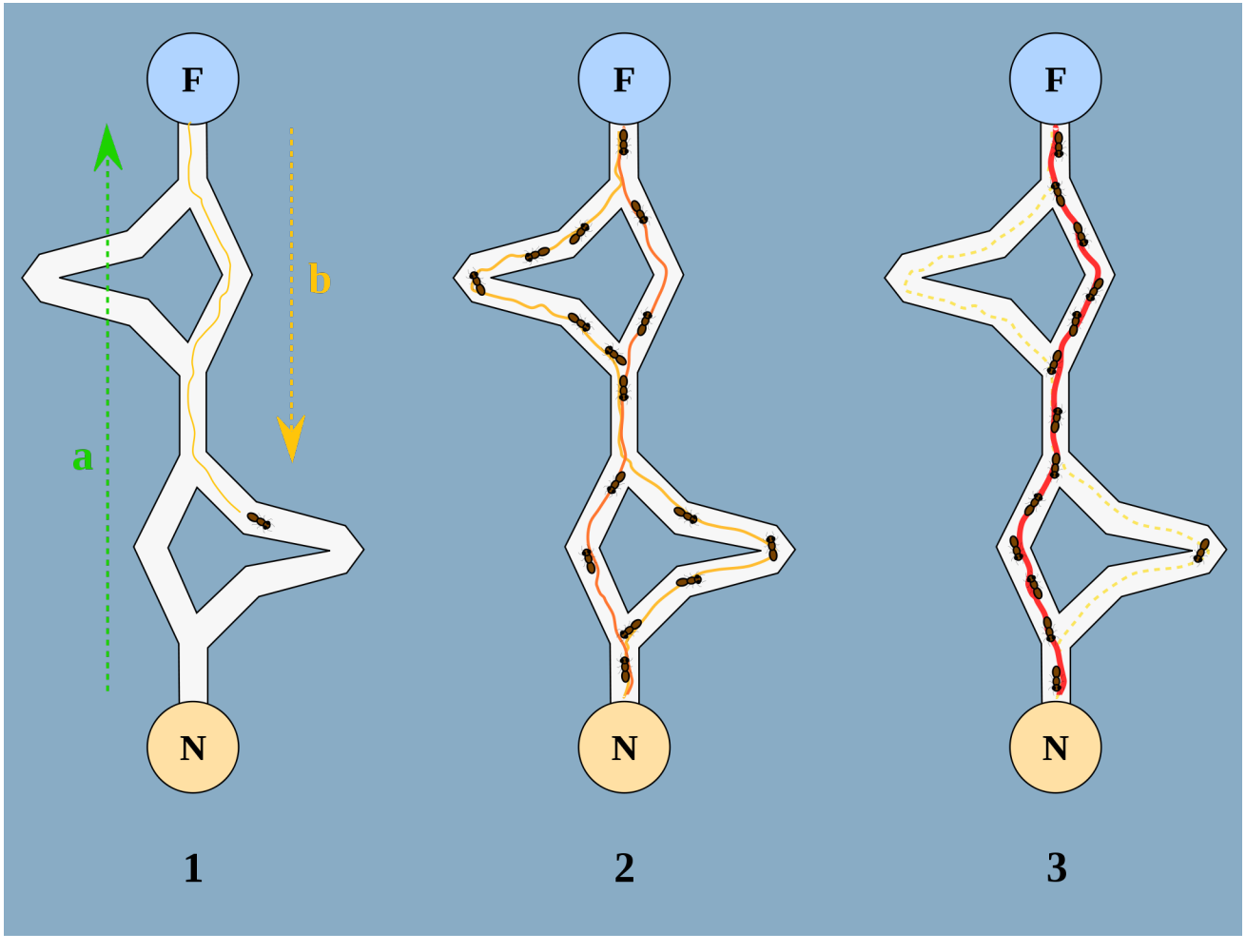
- Any problems?

Hand-raising assignment

- Any problems?
- Start node not part of the walk

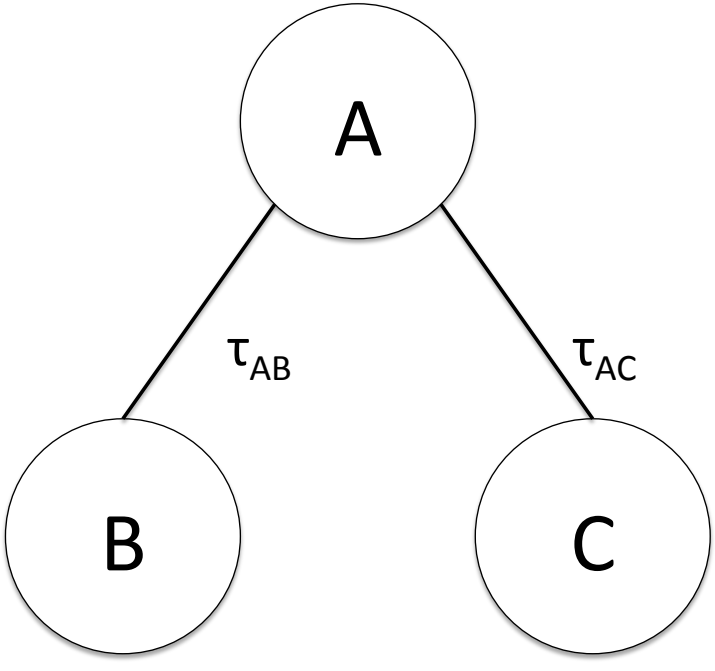
Ant Colony Optimisation—
Standard ACO

Standard ACO



Source of image: https://de.wikipedia.org/wiki/Ameisenalgorithmus/

ACO Pheromones



Pheromone update

$$\tau_{i,j} \leftarrow \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{8}$$

Standard ACO I

```
1  #Cost function
2  def getSum(edges):
3      return sum(e.cost for e in edges)
4  MAXCOST = getSum(edges)
5  class ANT:
6      def __init__(self):
7          self.visitedEdges = []
8
9      def walk(self,startNode):
10         currentNode = startNode
11         currentEdge = None
12         while(not checkAllNodesPresent(self.visitedEdges)):
13             currentEdge = currentNode.rouletteWheel(self.visitedEdges,↩
       startNode)
14             currentNode = currentEdge.toNode
```

Standard ACO II

```
15              self.visitedEdges.append(currentEdge)
16
17
18      def pheromones(self):
19          currentCost = getSum(self.visitedEdges)
20          if(currentCost<MAXCOST):
21              score = 1000**(1-float(currentCost)/MAXCOST) # Score ↩
            function
22              for oneEdge in self.visitedEdges:
23                  oneEdge.pheromones += score
24
25  for i in range(100000):
26      ant = ANT()
27      ant.walk(a)
28      ant.pheromones()
29      print i,getSum(ant.visitedEdges)
```

Standard ACO III

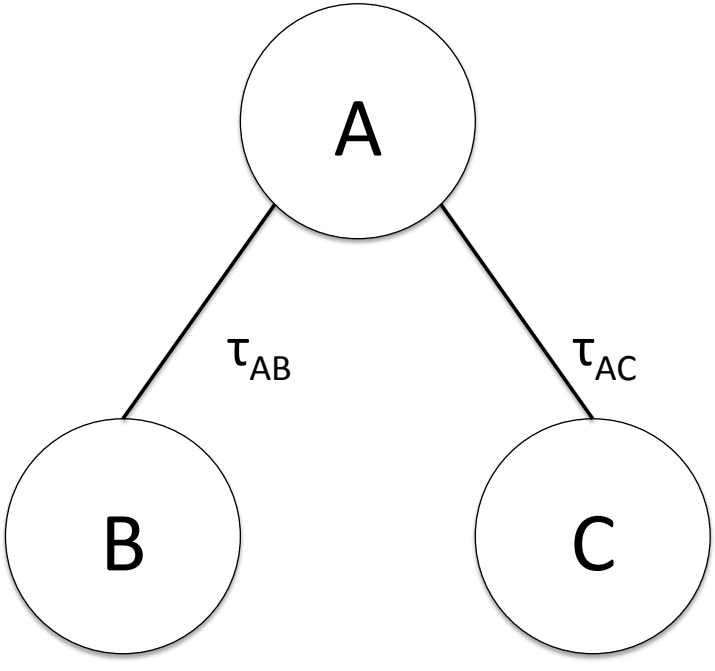
```
30
31  #Printing
32  ant = ANT()
33  ant.walk(a)
34  for edge in ant.visitedEdges:
35      print edge,edge.pheromones
```

Pheromone Intensity

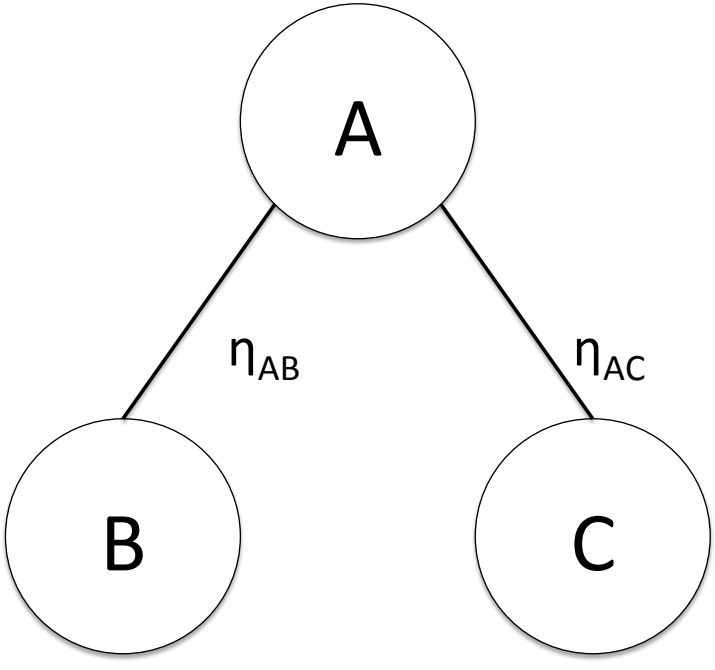
In an edge from Node i to j.

$$\tau_{ij} \tag{9}$$

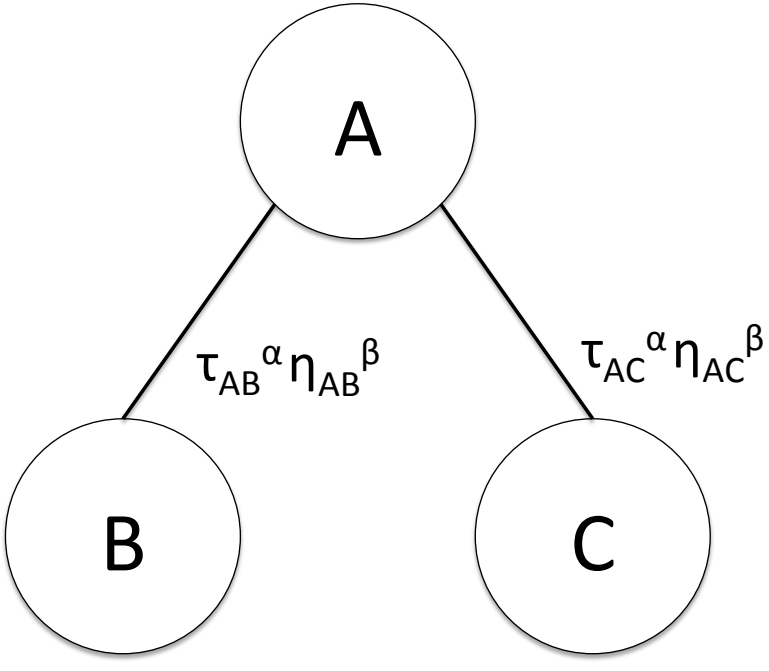
ACO Pheromones



ACO Cost



ACO Pheromones and Cost



Ant Transition Rule

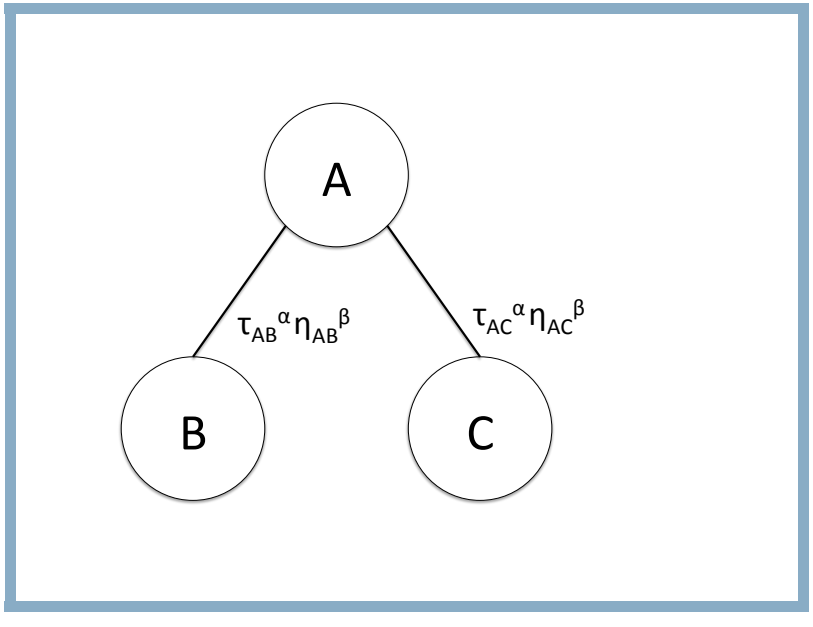
$$p_{i,j}^{k} = \begin{cases} \dfrac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{e_{i,l}} \tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}} & \text{if } e_{i,j} \in N(s^{p}) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

- $\tau_{i,j}$ pheromones
- $\eta_{i,j}$ cost
- $\alpha = 0$ greedy
- $\beta = 0$ rapid convergence

Ant Transition Rule, beta to zero, alpha to 1

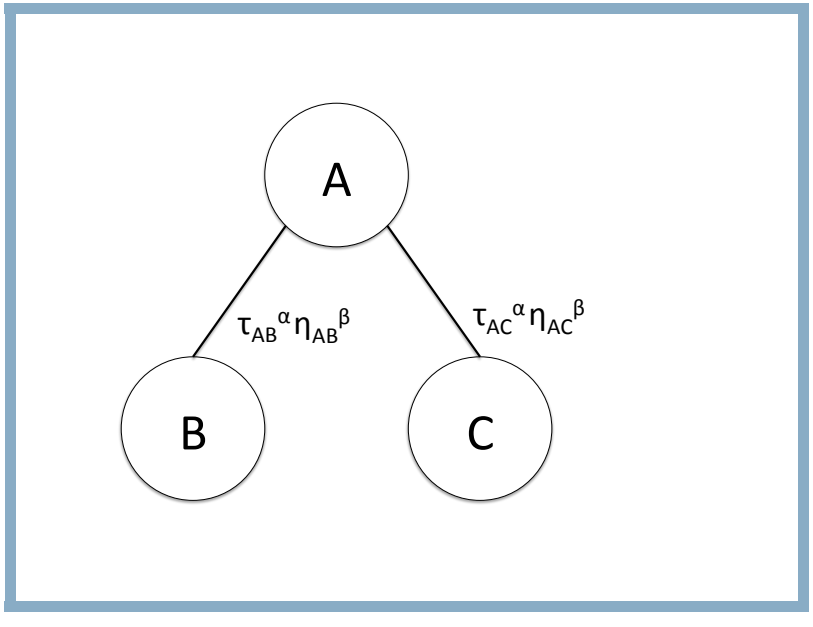
$$p_{i,j}^k = \begin{cases} \frac{\tau_{i,j}}{\sum_{e_{i,l}} \tau_{i,j}} & \text{if } e_{i,j} \in N(s^p) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Small assginment



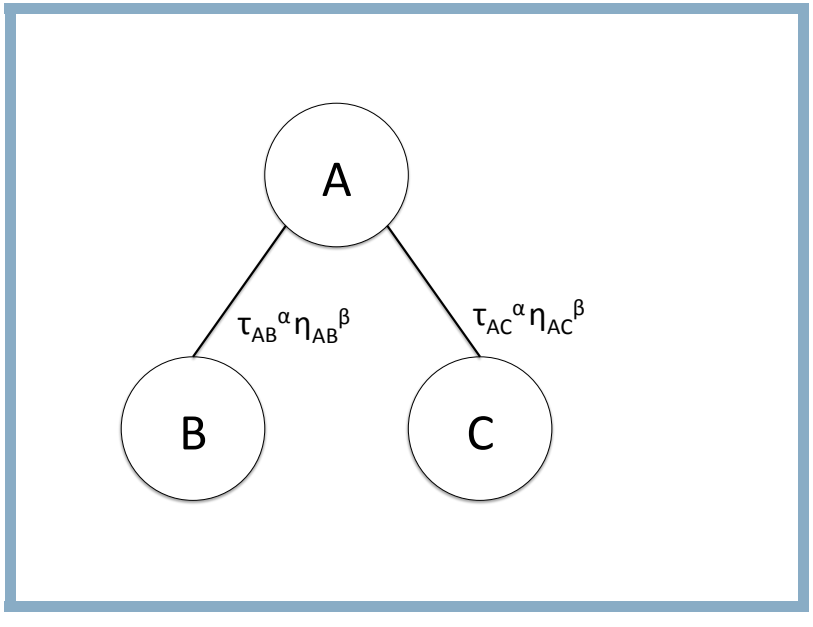
- $\alpha = 1$
- $\beta = 0$
- $\tau_{A,B} = 0.75$
- $\tau_{A,C} = 0.25$
- What is the probability to go from A to B.

Small assginment



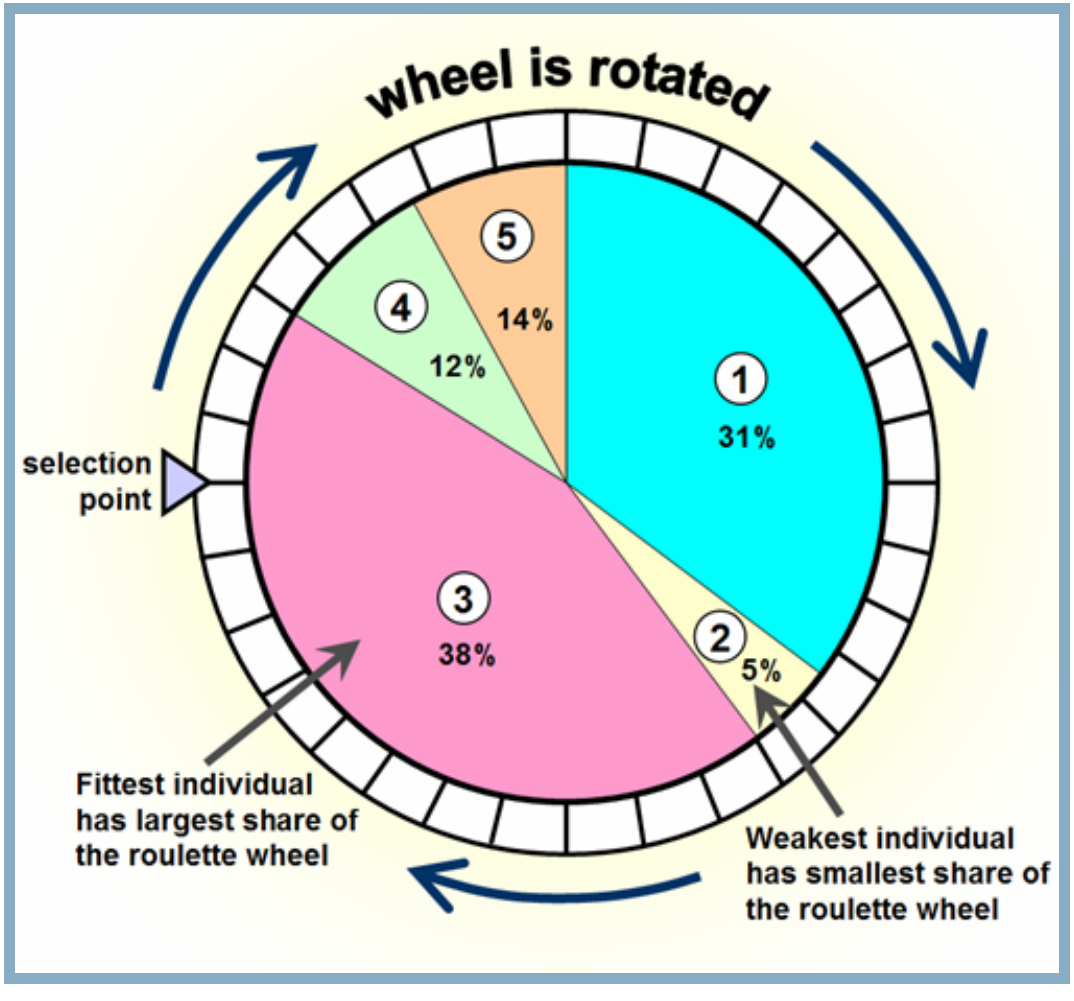
- $\alpha = 1$
- $\beta = 0$
- $\tau_{A,B} = 0.75$
- $\tau_{A,C} = 0.25$
- What is the probability to go from A to B.
- $\frac{\tau_{A,B}}{\tau_{A,B}+\tau_{A,C}} = \frac{0.75}{0.25+0.75} = 0.75$

Small assginment



- $\alpha = 1$
- $\beta = 0$
- $\tau_{A,B} = 0.75$
- $\tau_{A,C} = 0.25$
- What is the probability to go from A to B.
- $\frac{\tau_{A,B}}{\tau_{A,B}+\tau_{A,C}} = \frac{0.75}{0.25+0.75} = 0.75$
- Suggestions on how to implement this?

Roulette wheel selection



Source of image: `http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php`

Roulette wheel selection I

```
1   def rouletteWheel(self,visitedEdges,startNode):
2       visitedNodes = [oneEdge.toNode for oneEdge in visitedEdges]
3       viableEdges = [oneEdge for oneEdge in self.edges if not oneEdge.↩
    toNode in visitedNodes and oneEdge.toNode!=startNode]
4       if not viableEdges:
5           viableEdges = [oneEdge for oneEdge in self.edges if not ↩
    oneEdge.toNode in visitedNodes]
6
7       allPheromones = sum([oneEdge.pheromones for oneEdge in ↩
    viableEdges])
8       num = random.uniform(0,allPheromones)
9       s = 0
10      i = 0
11      selectedEdge = viableEdges[i]
12      while(s<=num):
```

Roulette wheel selection II

```
13              selectedEdge = viableEdges[i]
14              s += selectedEdge.pheromones
15              i += 1
16          return selectedEdge
```

Ant Colony Optimisation—
ACO With Evaporation

With Evaporation I

```
1  import random
2  MAXPHEROMONES = 100000
3  MINPHEROMONES = 1
4  #...
5  class Edge:
6      def __init__(self,fromNode,toNode,cost):
7          self.fromNode = fromNode
8          self.toNode = toNode
9          self.cost = cost
10         self.pheromones = MAXPHEROMONES
11
12     def checkPheromones(self):
13         if(self.pheromones>MAXPHEROMONES):
14             self.pheromones = MAXPHEROMONES
15         if(self.pheromones<MINPHEROMONES):
```

With Evaporation II

```
16              self.pheromones = MINPHEROMONES
17
18  def evaporate(edges):
19      for edge in edges:
20          edge.pheromones *= 0.99
21
22  def checkAllEdges(edges):
23      for edge in edges:
24          edge.checkPheromones()
25
26  for i in range(100000):
27      evaporate(edges)
28      ant = ANT()
29      ant.walk(a)
30      ant.pheromones()
31      checkAllEdges(edges)
```

With Evaporation III

```
32      print i,getSum(ant.visitedEdges)
```

Pheromone update with evaporation

$$\tau_{i,j} \leftarrow (1-p)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{12}$$

Ant Colony Optimisation—
MMAS

MinMax Ant colony System

- Pheromones have a MAX and MIN.
- Pheromones start at MAX
- Only release pheromones on the solution with the lowest cost.

Min Max ACO I

```
1  bestScore = 0
2  bestSolution = []
3  class ANT:
4  #...
5      def pheromones(self):
6          currentCost = getSum(self.visitedEdges)
7          if(currentCost<MAXCOST):
8              score = 1000**(1-float(currentCost)/MAXCOST) # Score ↩
           function
9              global bestScore
10             global bestEdges
11             if(score>bestScore):
12                 bestScore = score
13                 bestEdges = self.visitedEdges
14             for oneEdge in bestEdges:
15                 oneEdge.pheromones += score
```

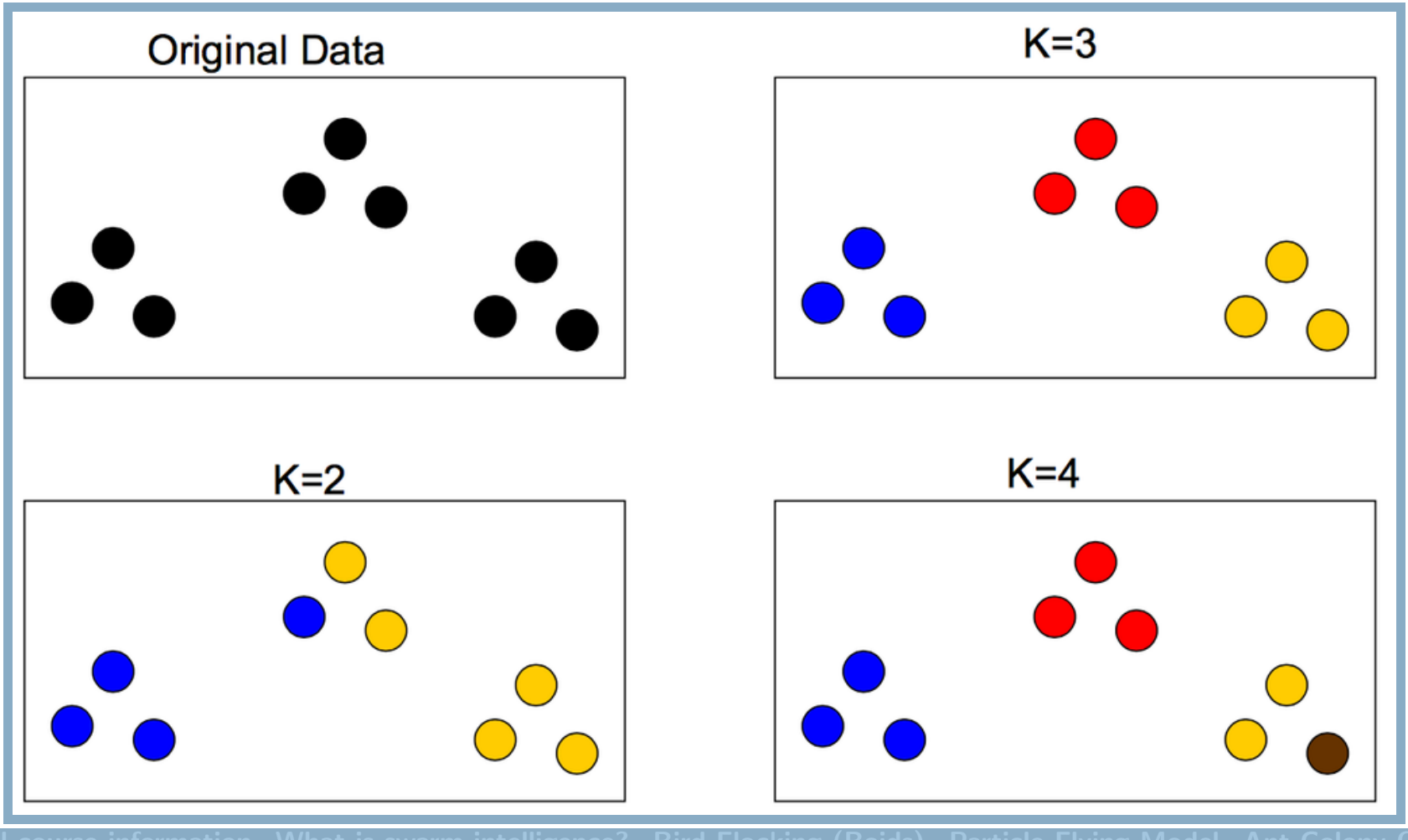
Min Max ACO II

Multilevel ACO

Multilevel ACO—
Clustering

Clustering

- K-means clustering



K-means Pseudo code

```
1 Pick K random points
2 Assign data to the closest points
3 Calculate cluster means
4 Go to 2
5 Stop when no point change clusers
```

Clustering I

```
1  import math
2
3  class Node:
4      def __init__(self,x,y,cluster):
5          self.x = x
6          self.y = y
7          self.cluster = cluster
8
9
10 NCLUSTERS = 3
11
12 data = [Node(random.uniform(0,100),random.uniform(0,100),random.↩
       randint(0,NCLUSTERS-1)) for i in range(1000)]
13
14 def getDistance(xone,yone,xtwo,ytwo):
```

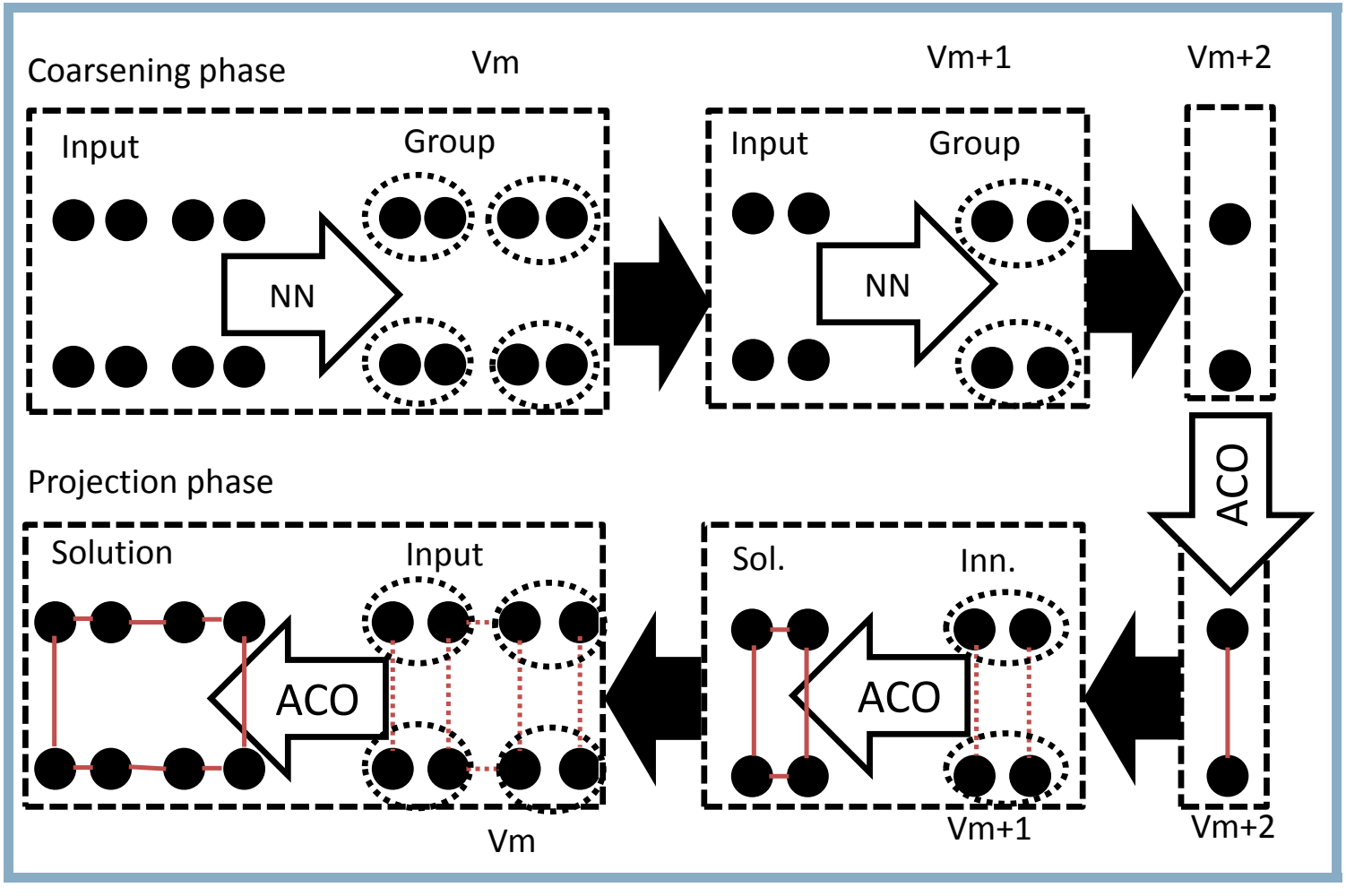
Clustering II

```
15      return abs(math.sqrt((xone-xtwo)**2 + (yone-ytwo)**2))
16
17  def kmeans():
18      global data
19      centroids = {}
20      for cluster in range(NCLUSTERS):
21          thisdata = [n for n in data if n.cluster==cluster]
22          avgx = sum([n.x for n in thisdata])/len(thisdata)
23          avgy = sum([n.y for n in thisdata])/len(thisdata)
24          print cluster,len(thisdata)
25          centroids[cluster] = (avgx,avgy)
26      print centroids
27
28      for node in data:
29          distance = 100000
30          for c,coordinate in centroids.items():
```

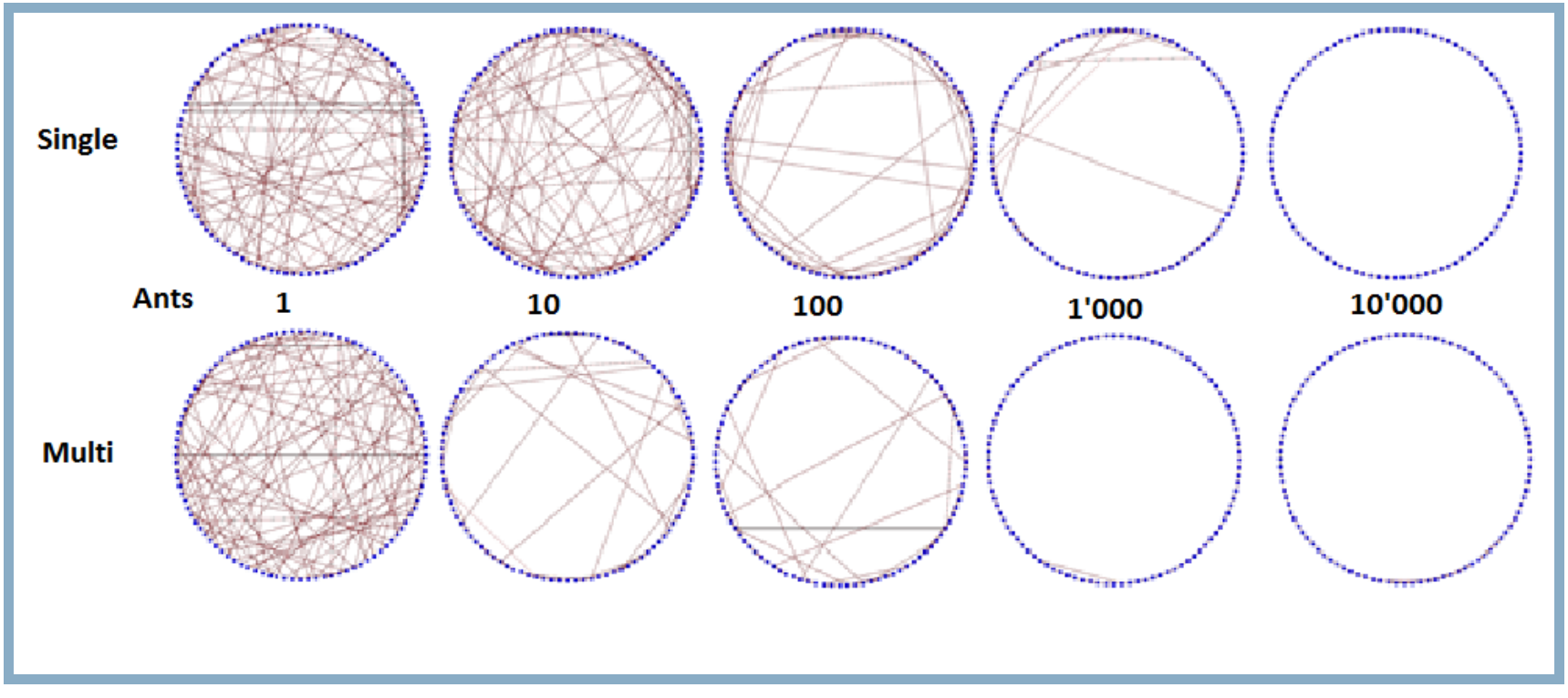
Clustering III

```
31              if(getDistance(node.x,node.y,coordinate[0],coordinate[1])<distance):
32                  closestcluster = c
33                  distance = getDistance(node.x,node.y,coordinate[0],coordinate[1])
34            node.cluster = closestcluster
35        return centroids
36
37
38    def getNextData():
39        centroids = kmeans()
40        return [(n.x,n.y) for n in data if n.cluster==0],[(n.x,n.y) for n in data if n.cluster==1],[(n.x,n.y) for n in data if n.cluster==2]
```

Multilevel

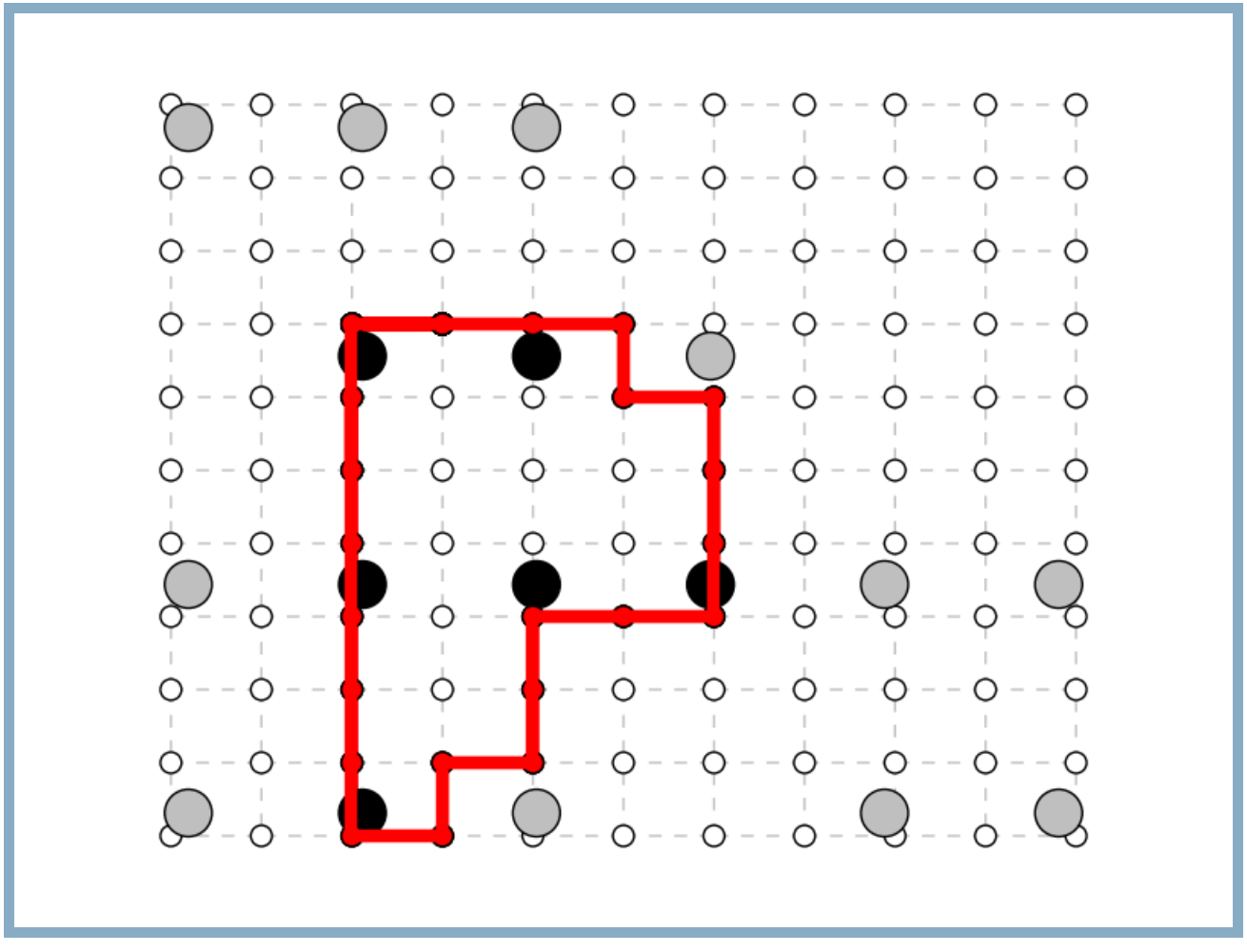

Coarsening phase
Vm
Vm+1
Vm+2
Input
Group
NN
Input
Group
NN
ACO
Projection phase
Solution
Input
ACO
Sol.
Inn.
ACO
Vm
Vm+1
Vm+2

Results

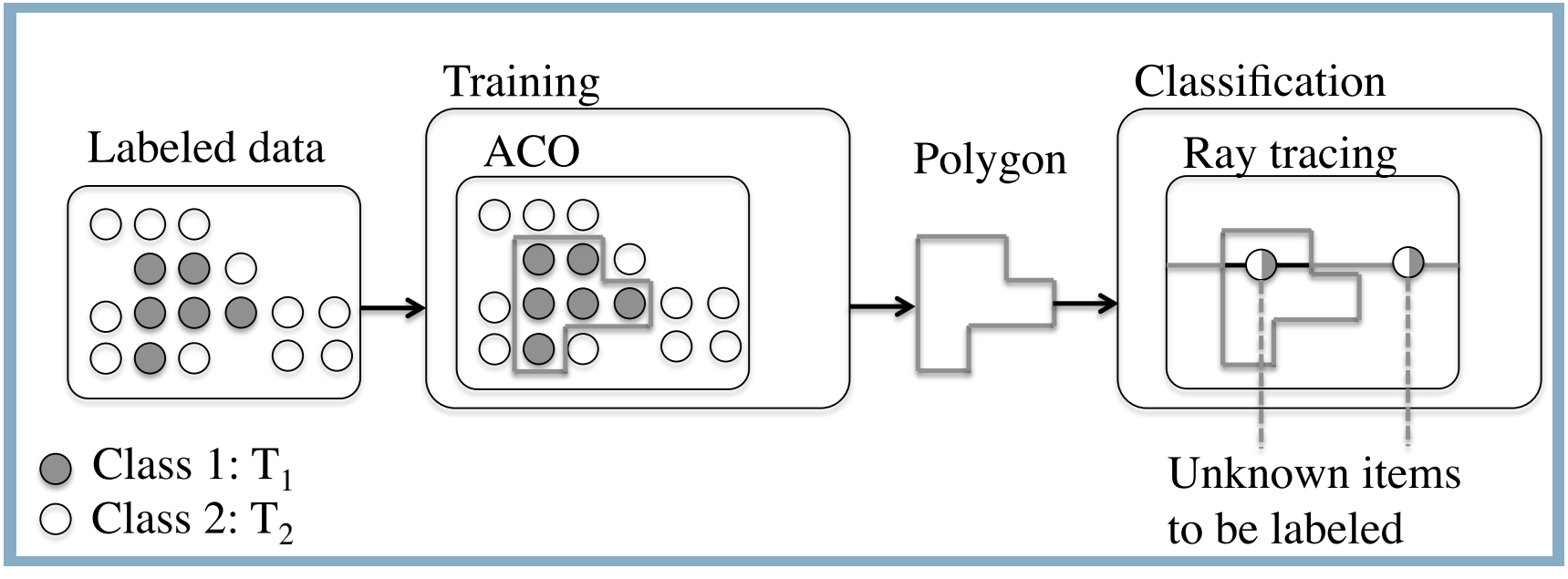


Classification using ACO— PolyACO

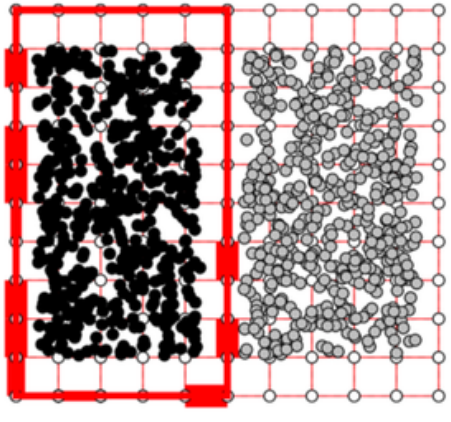
Problem



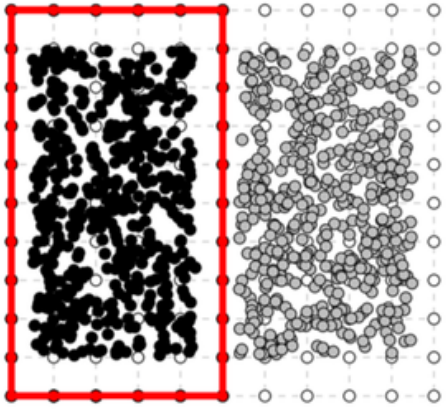
PolyACO


Training
Classification
Labeled data
ACO
Polygon
Ray tracing
Class 1: T_1
Class 2: T_2
Unknown items
to be labeled

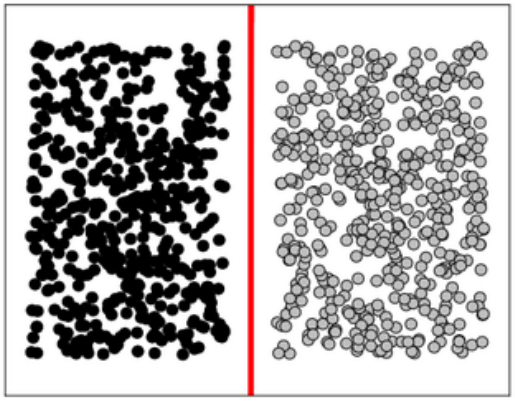
POC



a) PolyACO Pheromones

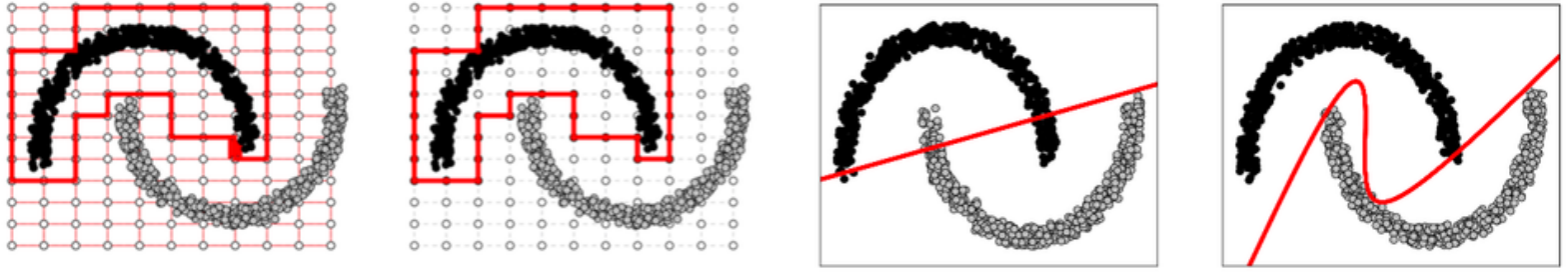


b) PolyACO Polygon Solution



c) Linear SVM

Half-moons



a) PolyACO Pheromones b) PolyACO Polygon Solution c) Linear SVM d) Polynomial SVM

Fig. 7. Example of solutions based on semi-circles with the classes Black (T_1) and Gray (T_2).

PolyACO over time

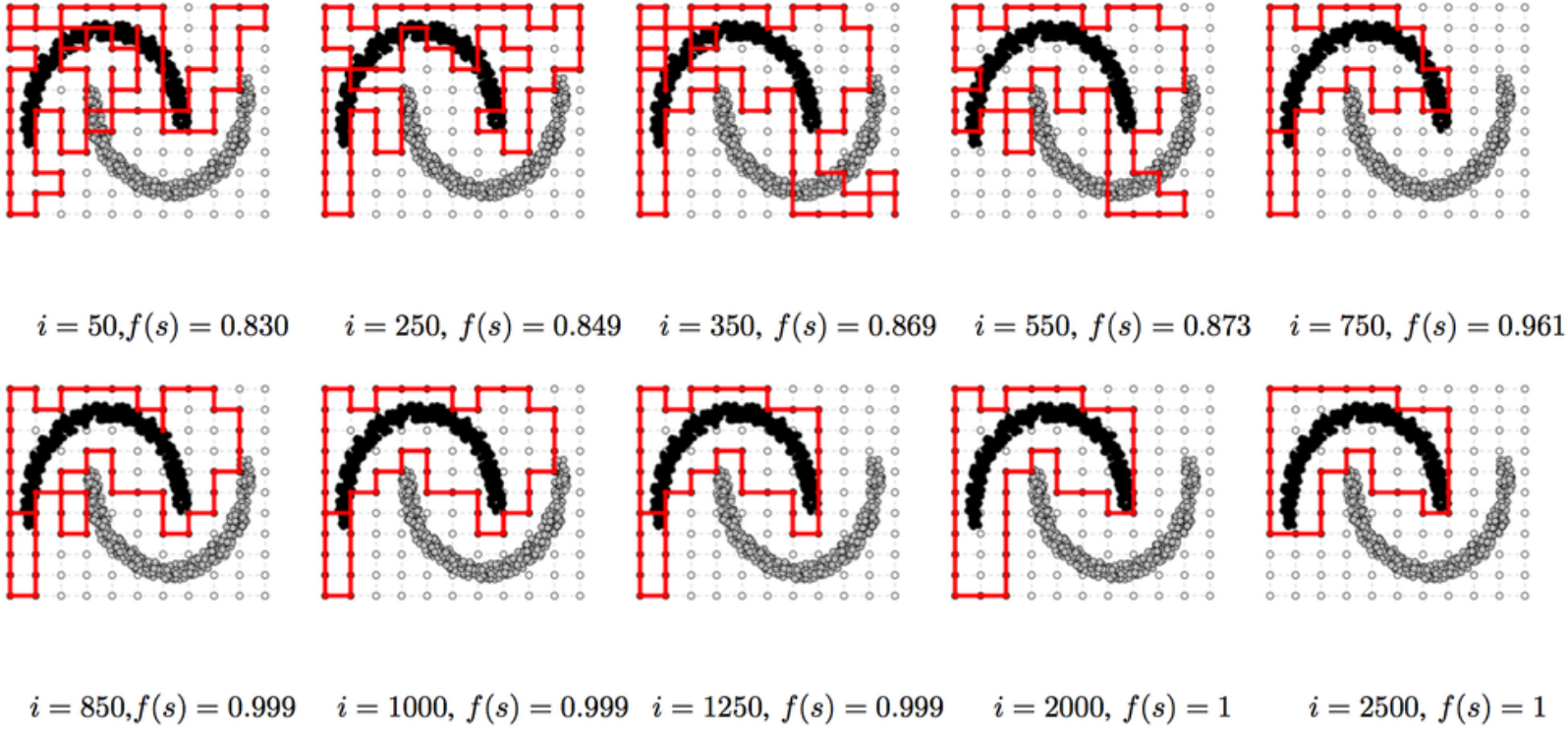
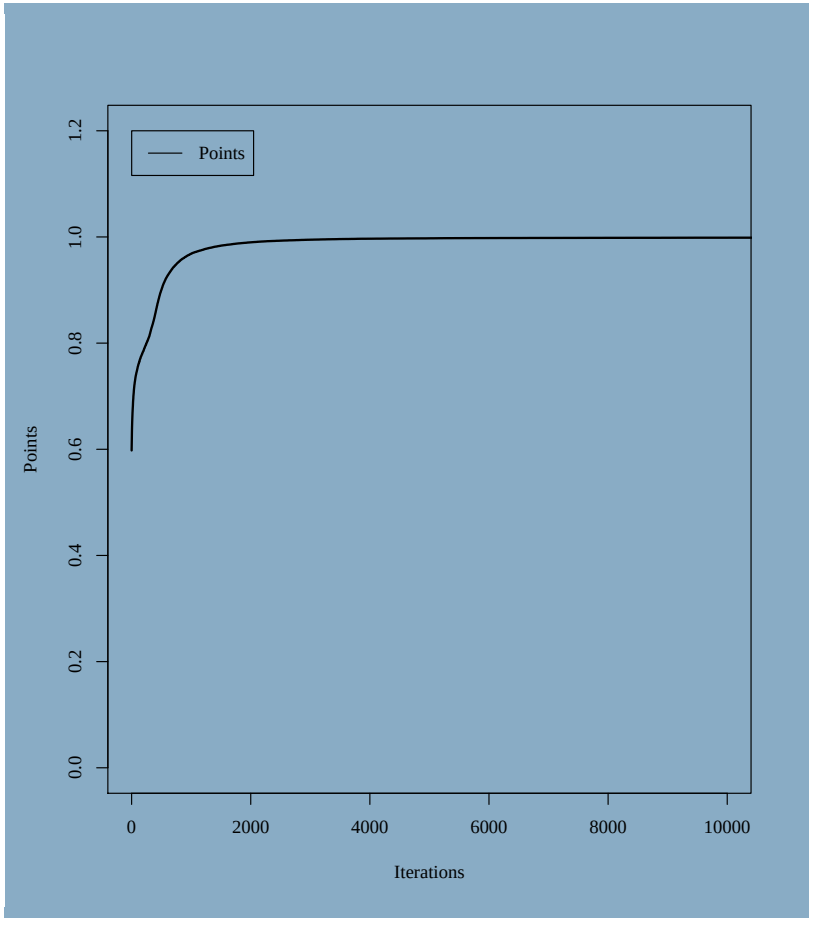
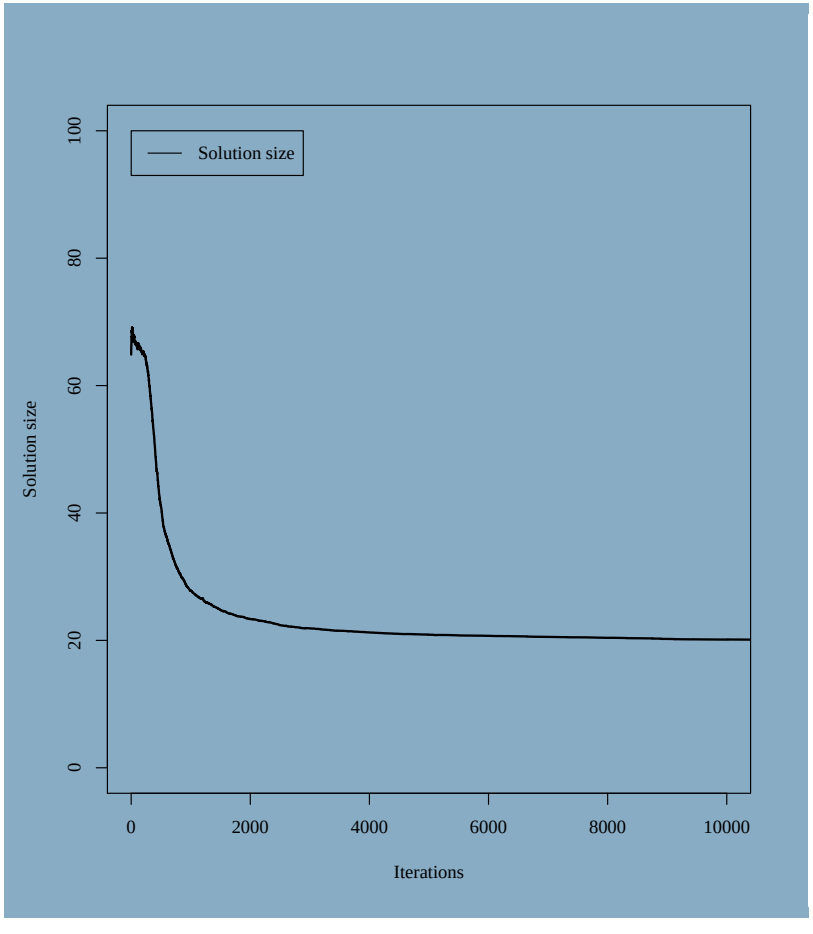


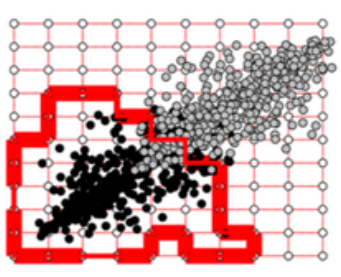
Fig. 3. Example of best known polygon, $s*$ over training periods.

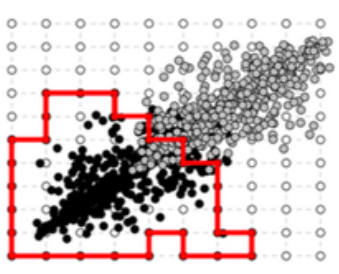
PolyACO over time

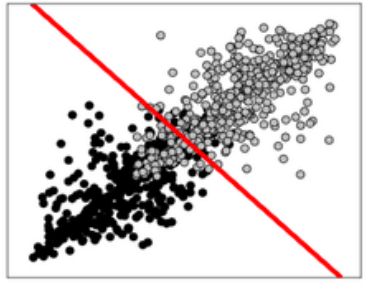


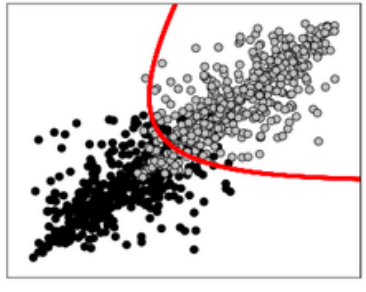


Noise



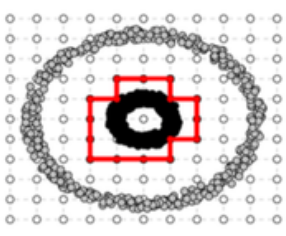




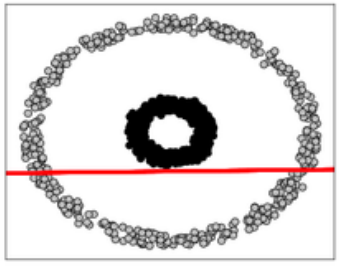


a) PolyACO Pheromones b) PolyACO Polygon Solution c) Linear SVM d) Polynomial SVM

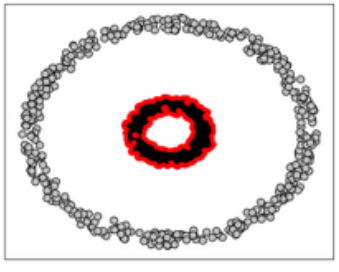
Circle



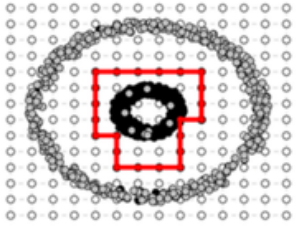
a) PolyACO Polygon Solution



b) Linear SVM



c) RBF SVM

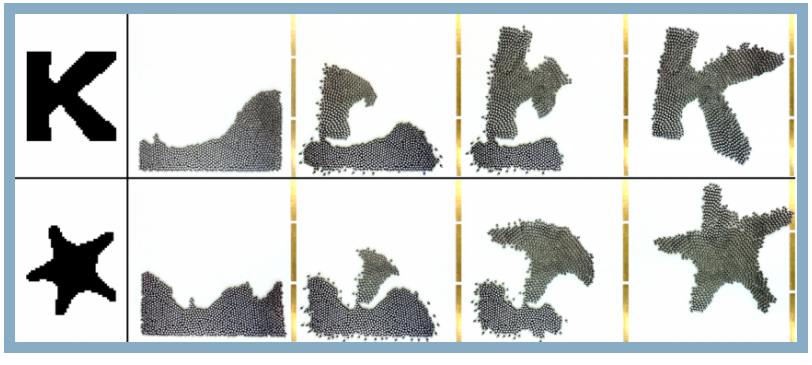


d) PolyACO Polygon Solution with 5% noise

Classification using ACO— PolyACO—

Practical Usages

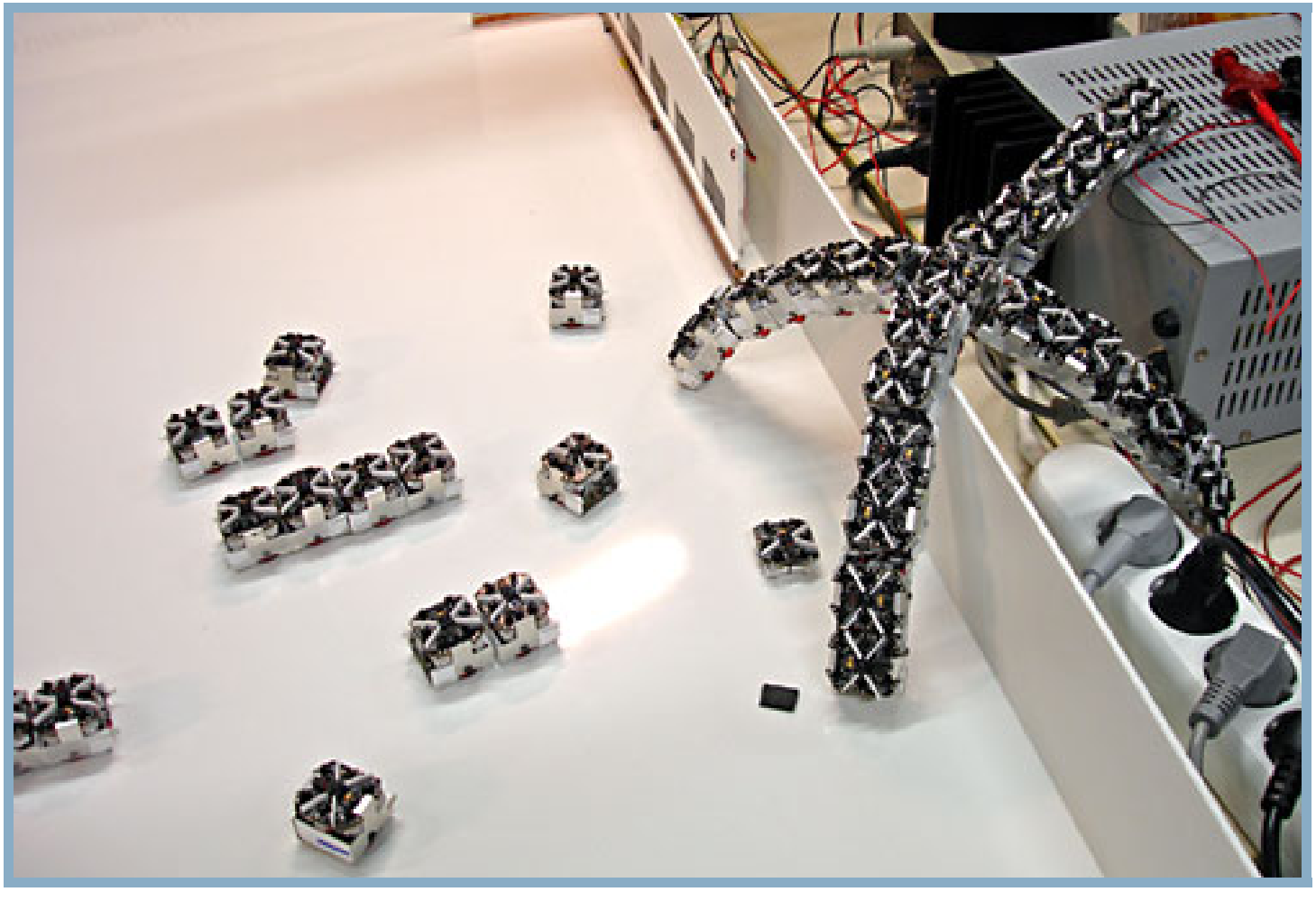
Building



Source of image: http://phys.org/news/2014-08-autonomous-robots-self-organizing-thousand-r html

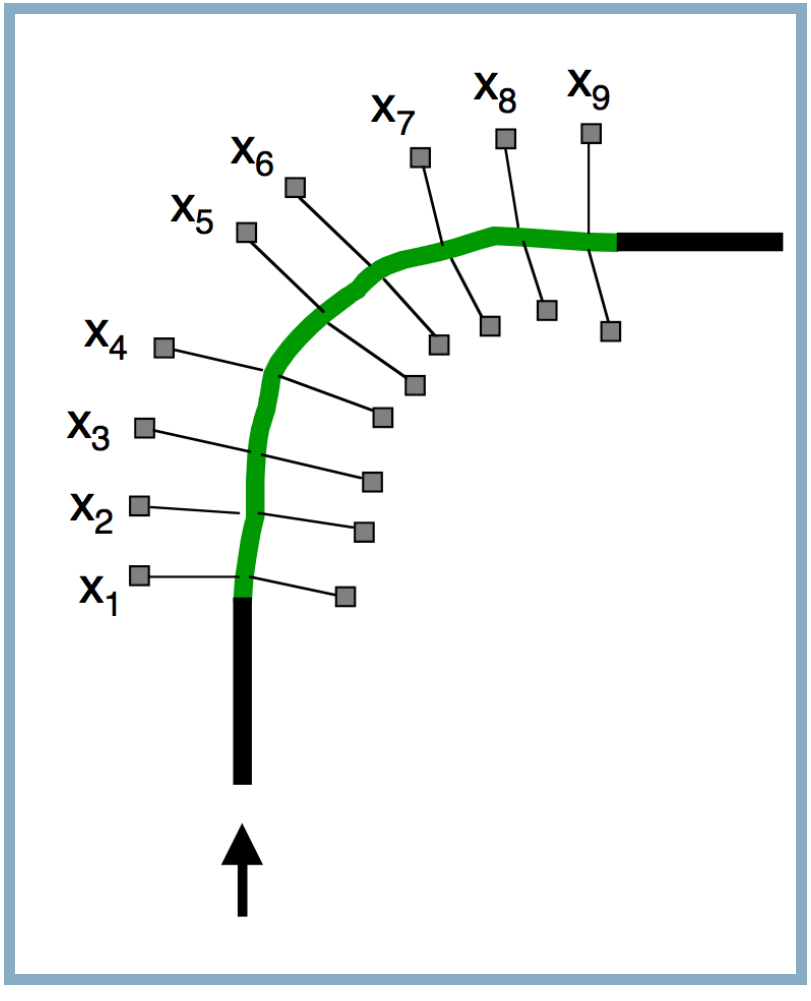
Source of image:

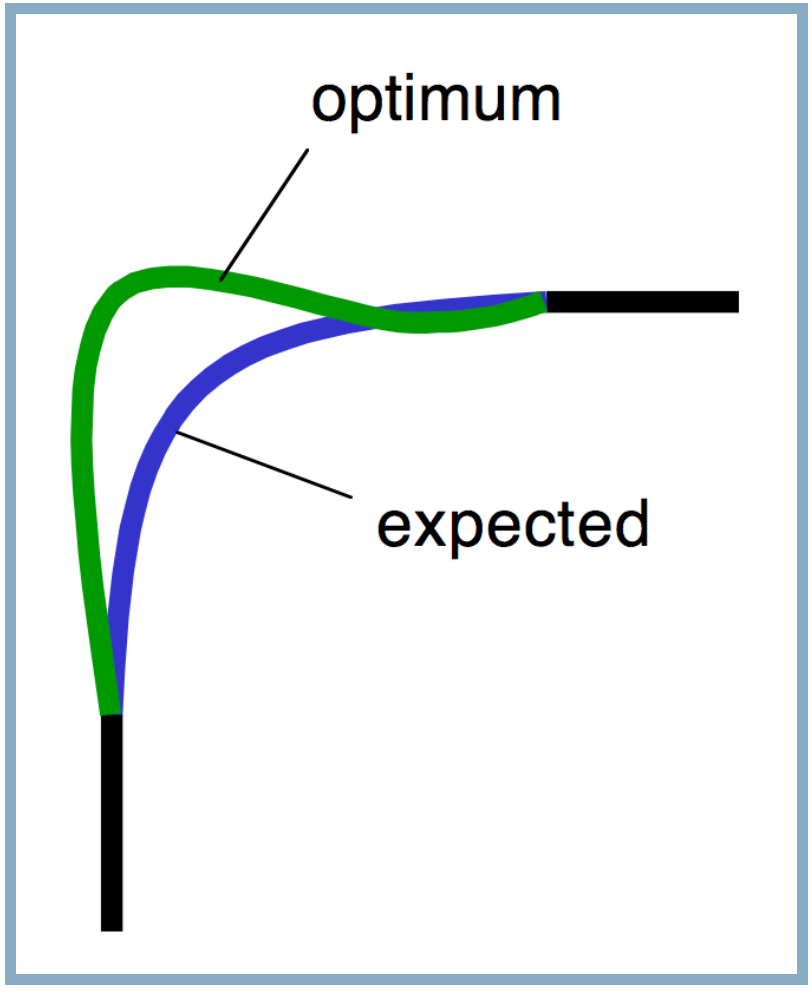
Building 2



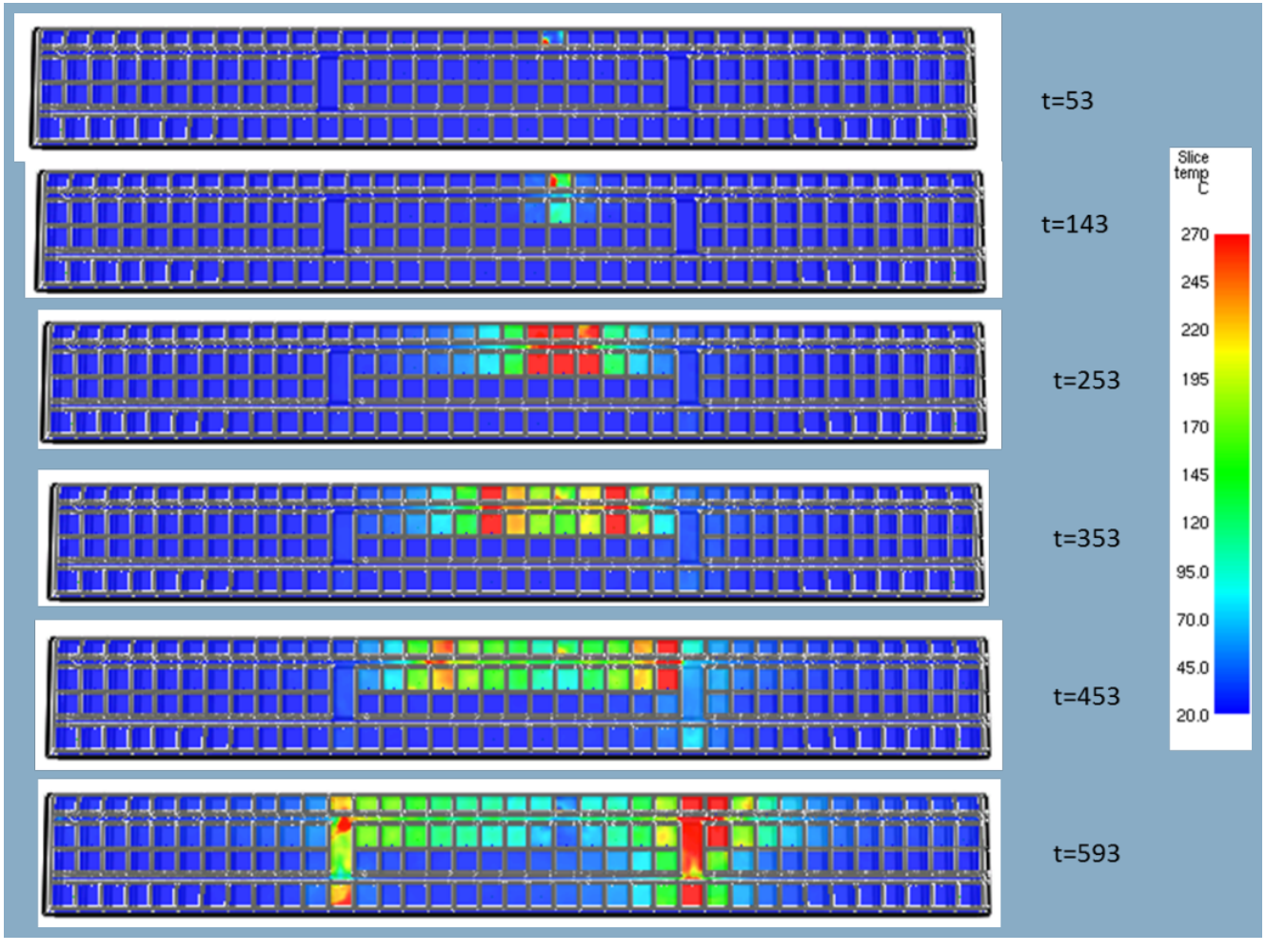
Source of image: http://coolmoro2.tumblr.com

Maximum water flow



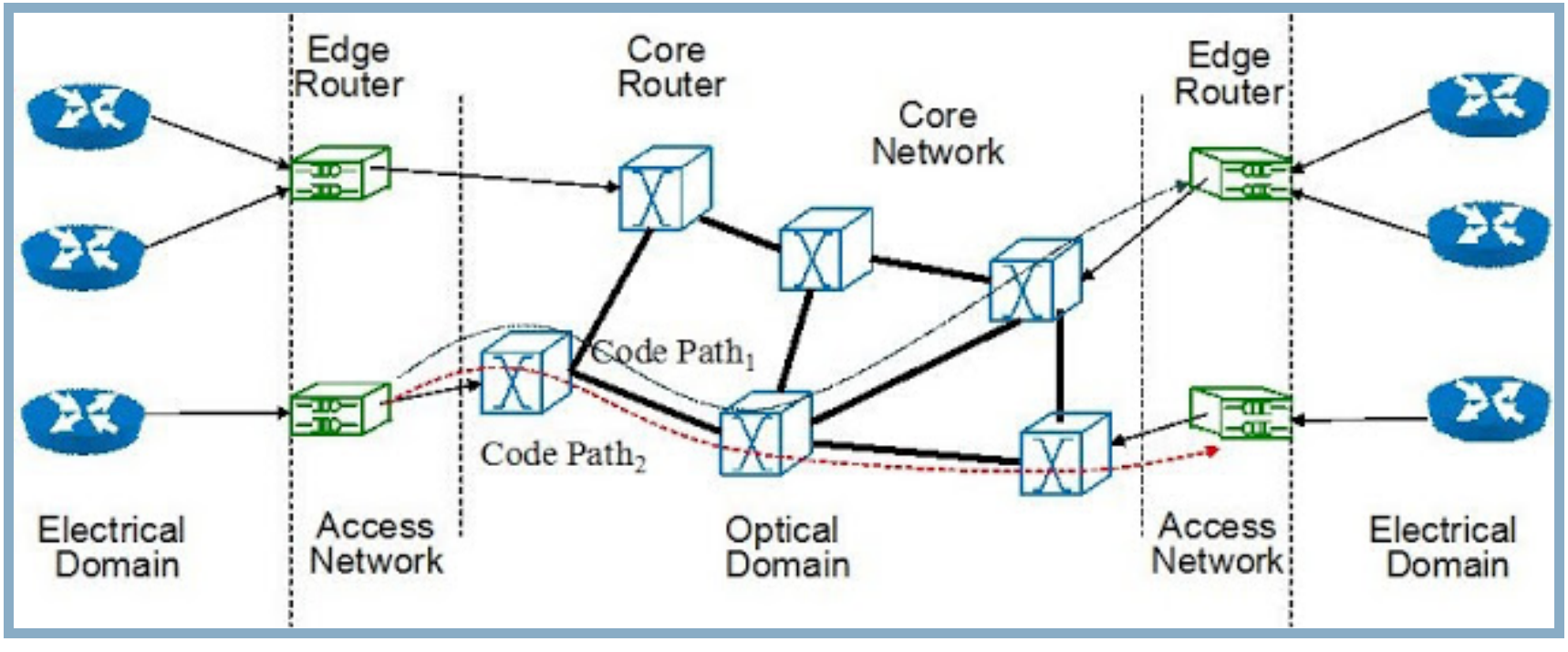


Escape planning



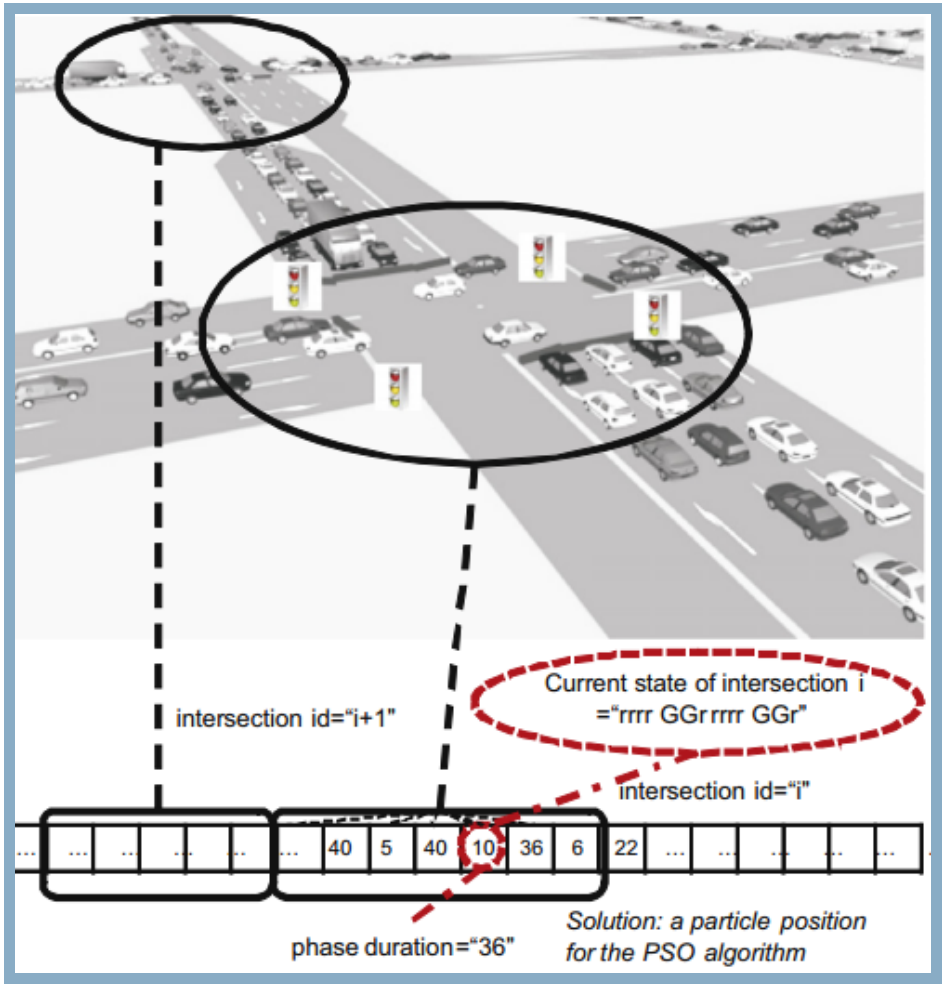
[Goodwin et al., 2015]

Network optimisation



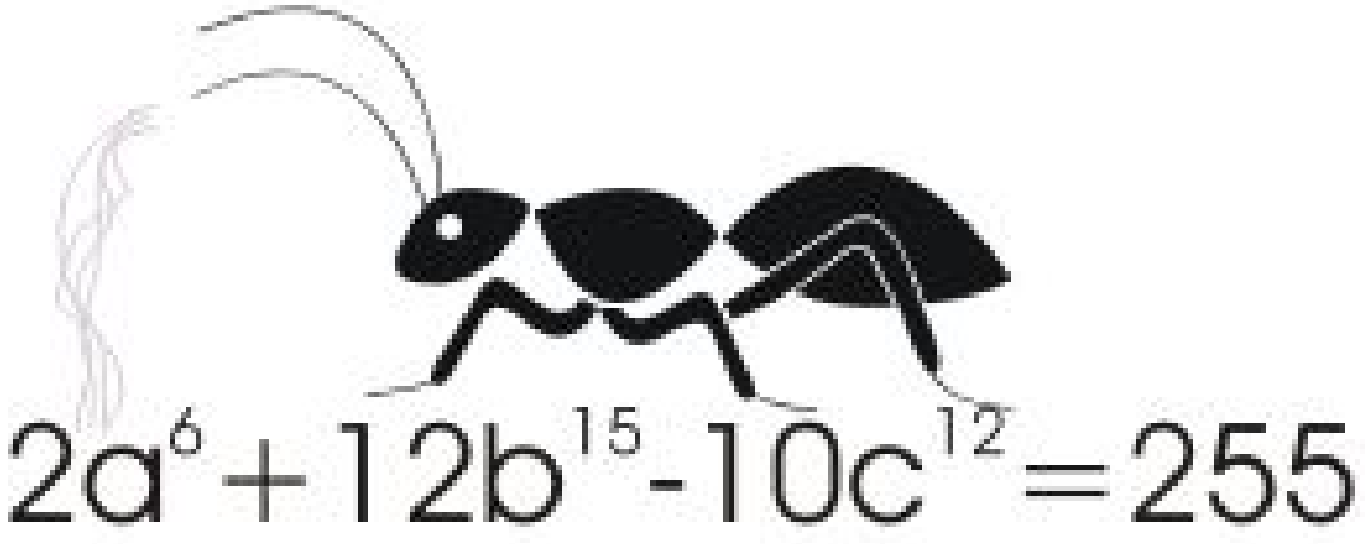
Source of image: `http://www.techferry.com/articles/swarm-intelligence.html`

Traffic intersection planning



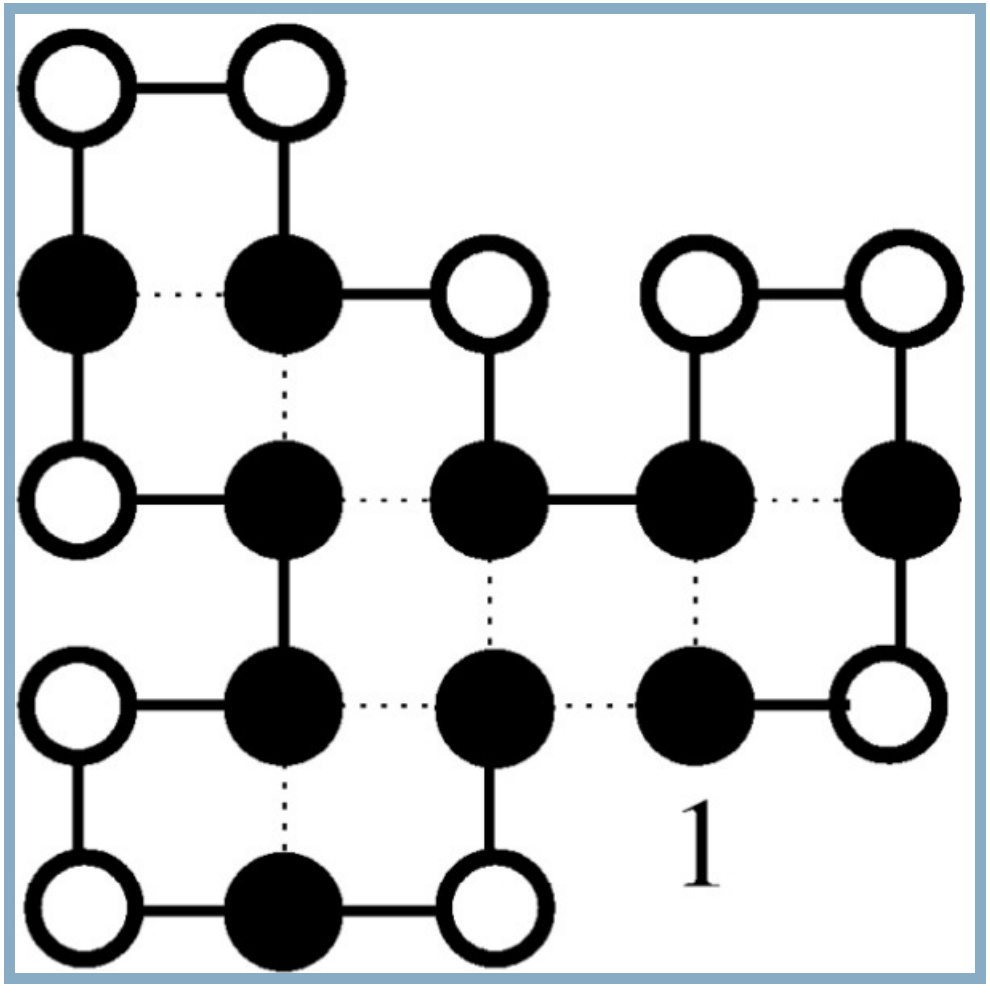
Source of image: http://www.techferry.com/articles/swarm-intelligence.html

Regression,solving equations



Source of image: http://www.i-programmer.info/news/112-theory/

Protein folding

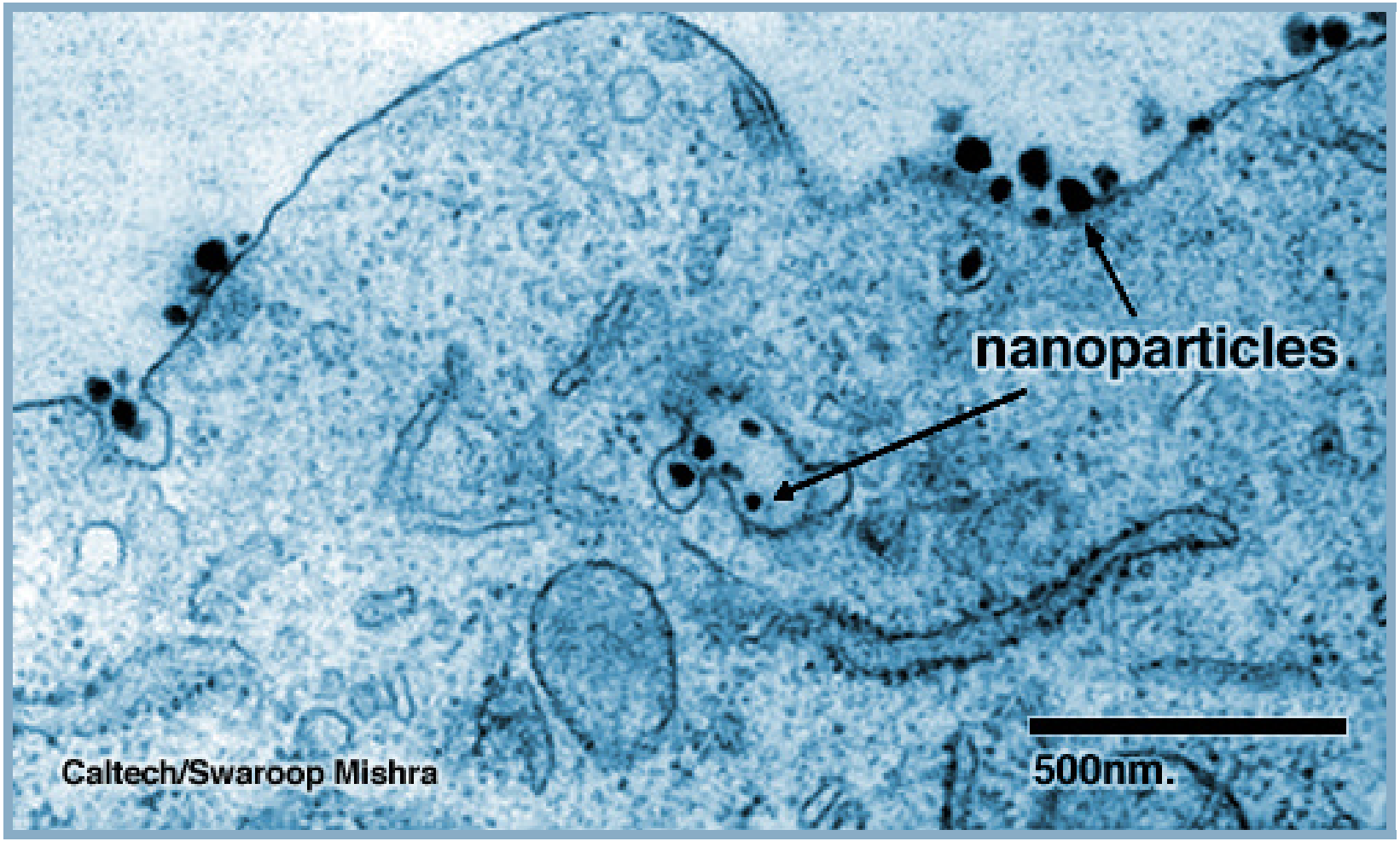


Source of image: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-6-30

Feature selection

Source of image: https://www.researchgate.net/publication/225156352_Feature_Selection_Using_Ant_Colony_Optimization_ACO_A_New_Method_and_Comparative_Study_In_the_Application_of_Face_Recognition_System

Cancer Nanobots



Source of image: http://www.zmescience.com/research/photograph-nanobots-cancer-25032010/

[Cerofolini and Amato, 2013][Lewis and Bekey, 1992]

Philosphical

- Individual ants have no intelligence.
- The group has some intelligence.
- Where does the intelligence come from?

Assignment

Manadotry Assignment

- Download the World Cities Database: `http://download.maxmind.com/download/worldcities/worldcitiespop.txt.gz`
- A backpacker should travel from a (1) Start city to a (2) Destination city, (3) visiting at least N number of cities. Which cities are not important.
- Example: Travel from (1) no,oslo, to (2) no,bergen, (3) Visiting at least 10 cities on his way.
- Use the (any variant of ACO) to find the shortest path with the above criteria.
- Assumptions: The distance between two cities is the air distance.

Mandatory Assignemnt (2)

- Hints:
  - Remove duplicate cities.
  - Reduce the map to make the problem easier in the testing phase, e.g. only Norway.
- Reduce the problem if needed.
- Make necesarry assumptions to solve the task.
- How many cities can you support?
- Presentation in two weeks.

Resources

- Natural Computing, Lecture 10: Ant Colony Optimisation, Michael Herrmann
- Swarm Intelligence, From Natural to Artificial Systems, Ukradnute kde sa dalo
- Swarm Intelligence Introduction, Thiemo Krink,

References


Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999).
Swarm intelligence: from natural to artificial systems.
Number 1. Oxford university press.


Cerofolini, G. and Amato, P. (2013).
Sensing strategies for early diagnosis of cancer by swarm of nanorobots: An evidential paradigm.
In *Nanorobotics*, pages 331–352. Springer.


Goodwin, M., Granmo, O.-C., and Radianti, J. (2015).
Escape planning in realistic fire scenarios with ant colony optimisation.
Applied Intelligence, 42(1):24–35.


Lewis, M. A. and Bekey, G. A. (1992).
The behavioral self-organization of nanorobots using local