UNIVERSITY OF LOUISVILLE
J.B. SPEED SCHOOL OF ENGINEERING
Computer Engineering and Computer Science Department

# CECS545-Artificial Intelligence

## Constraint Satisfaction Problems & Swarm Intelligence

Dr. Roman Yampolskiy

# Constraint satisfaction problems (CSPs)

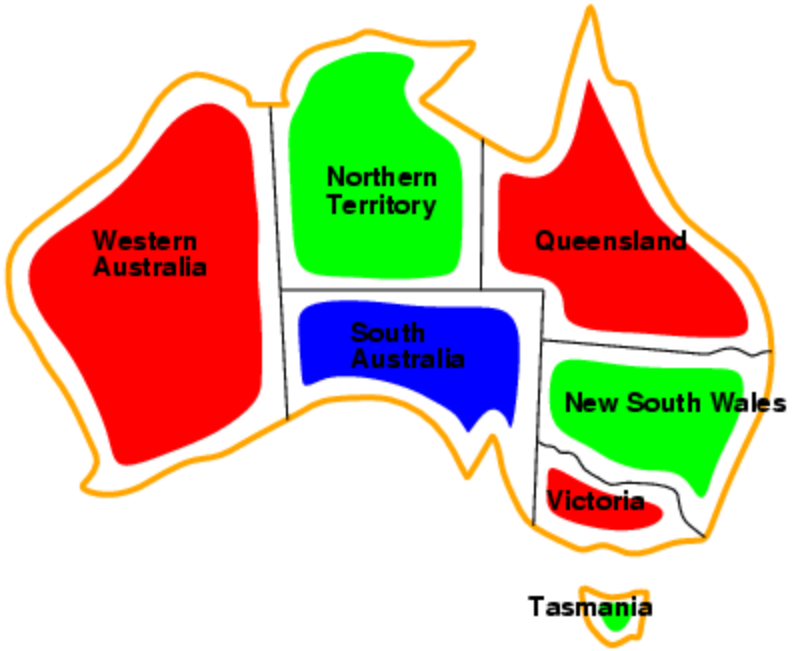
- CSP:
  - state is defined by variables $X_i$ with values from domain $D_i$
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Allows useful general-purpose algorithms with more power than standard search algorithms

# Example: Map-Coloring


- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains $D_i$ = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT

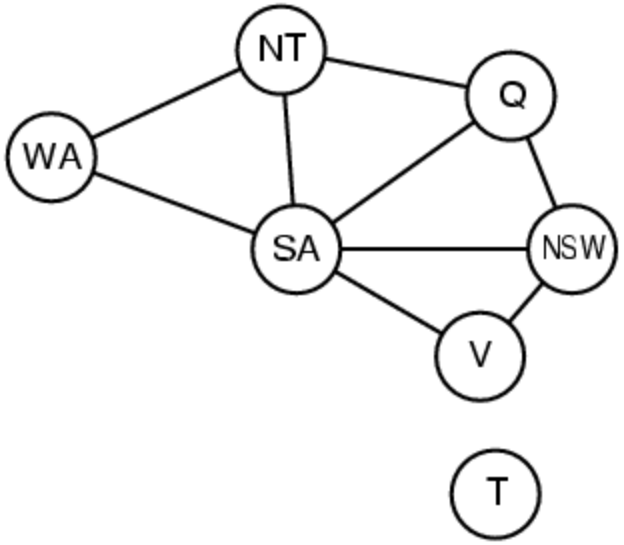
# Example: Map-Coloring




- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

# Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints

# Varieties of CSPs

- Discrete variables
  - finite domains:
    - $n$ variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - e.g., 3-SAT (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job:
      $StartJob_1 + 5 \leq StartJob_3$

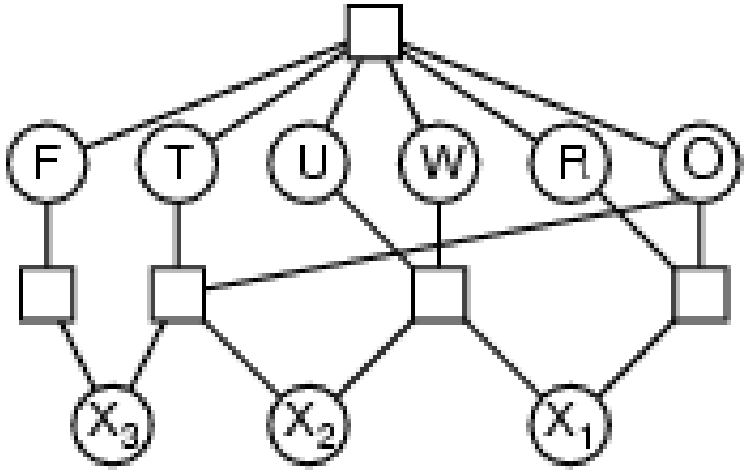
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

# Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables,
  - e.g., SA ≠ WA ≠ NT

# Example: Cryptarithmetic

$$\begin{array}{r} \phantom{+}\ T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



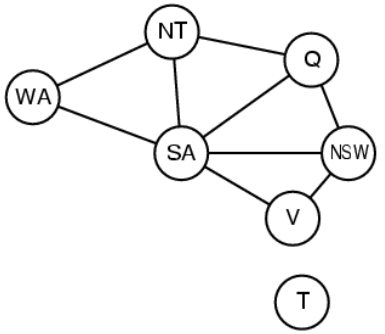

- Variables: *F T U W R O* $X_1\ X_2\ X_3$
- Domains: {*0,1,2,3,4,5,6,7,8,9*} {0,1}
- Constraints: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

# Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

- Notice that many real-world problems involve real-valued variables

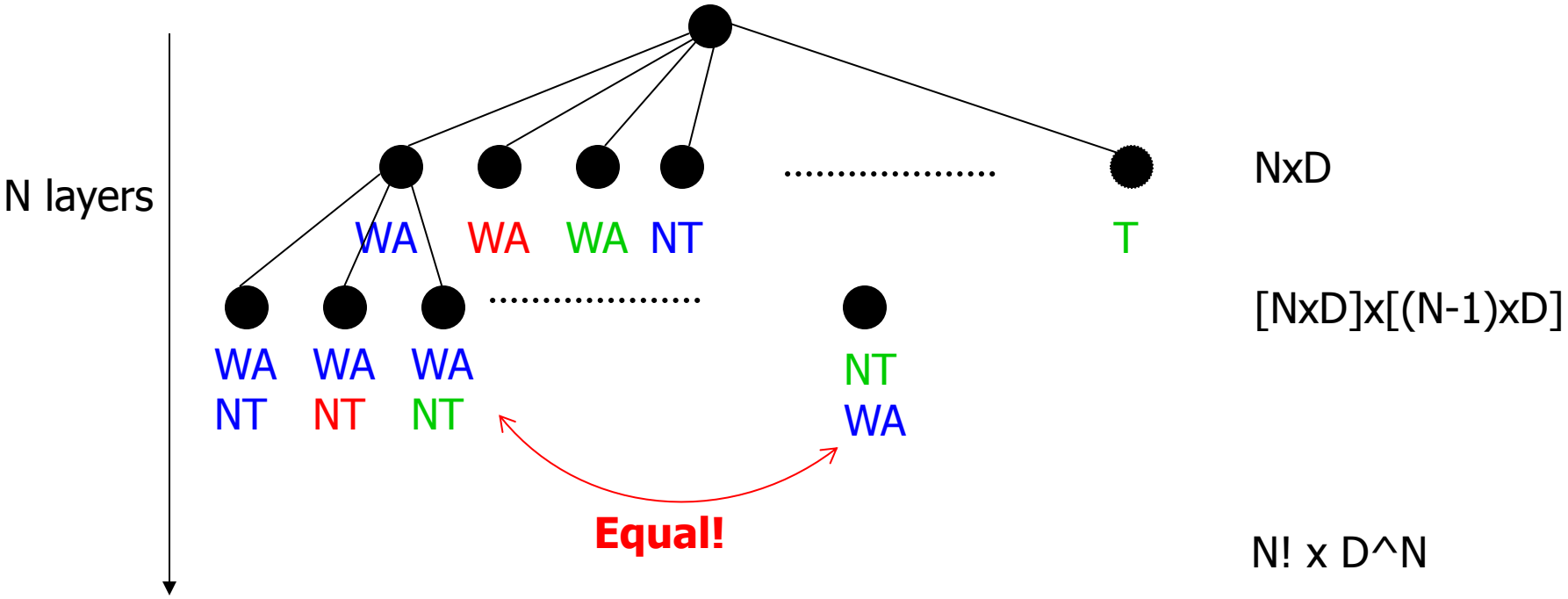
# Standard search formulation




Let's try the standard search formulation.

We need:

- Initial state: none of the variables has a value (color)
- Successor state: one of the variables without a value will get some value.
- Goal: all variables have a value and none of the constraints is violated.



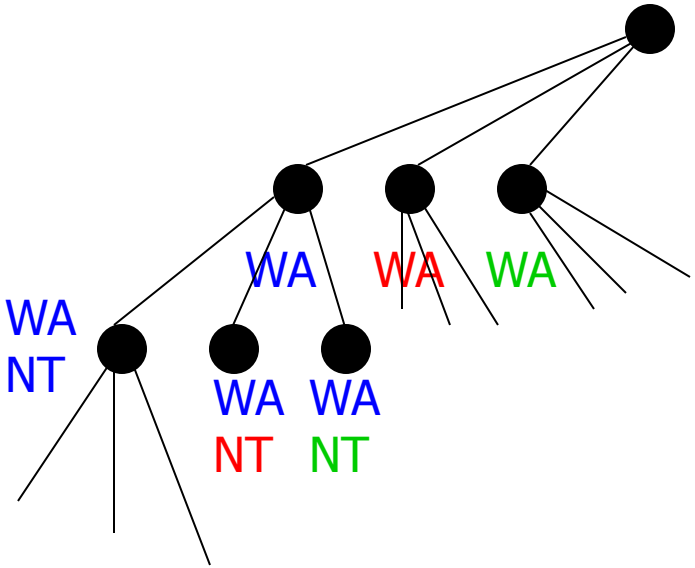

There are N! x D^N nodes in the tree but only D^N distinct states??

# Backtracking (Depth-First) search

- Special property of CSPs: They are commutative:
  This means: the order in which we assign variables does not matter.

NT WA = WA NT

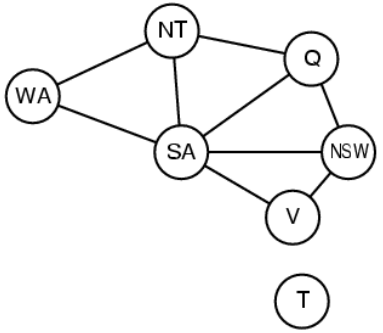
- Better search tree: First order variables, then assign them values one-by-one.




D

D^2

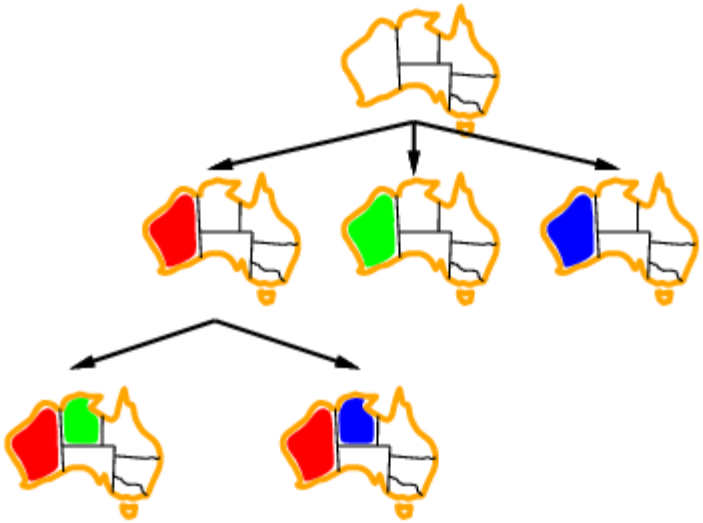
D^N



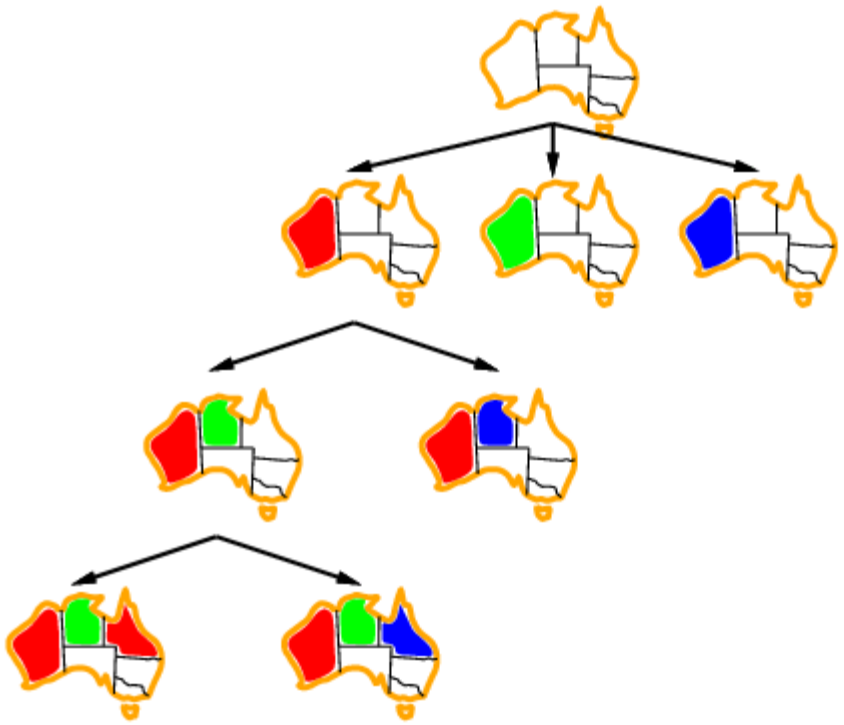
# Backtracking example

# Backtracking example

# Backtracking example



# Backtracking example



# Improving backtracking efficiency

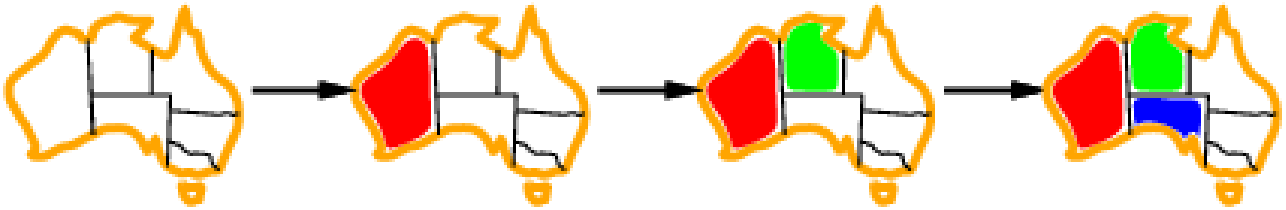
- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

  - We'll discuss heuristics for all these questions in the following.

# Which variable should be assigned next? →minimum remaining values heuristic

- Most constrained variable:

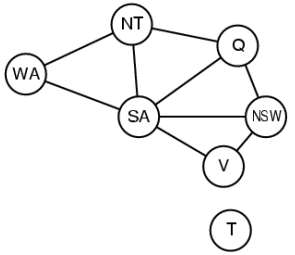
  choose the variable with the fewest legal values



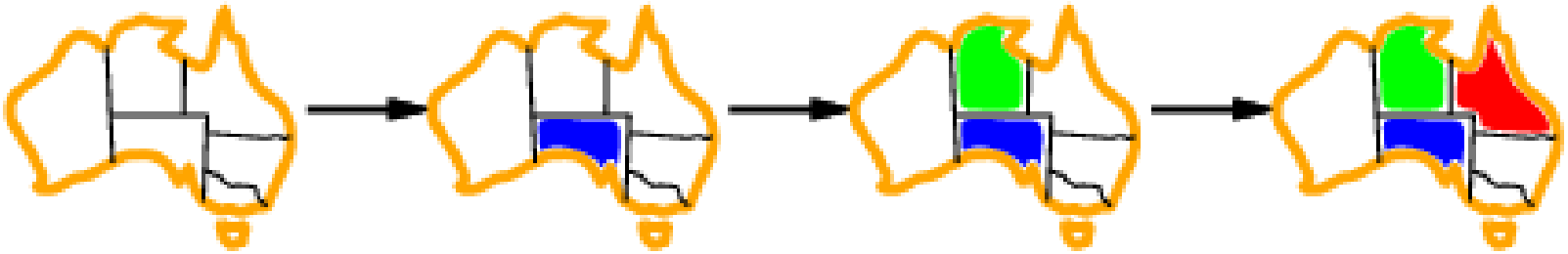
- a.k.a. minimum remaining values (MRV) heuristic
- Picks a variable which will cause failure as soon as possible, allowing the tree to be pruned.

# Which variable should be assigned next? → degree heuristic

- Tie-breaker among most constrained variables

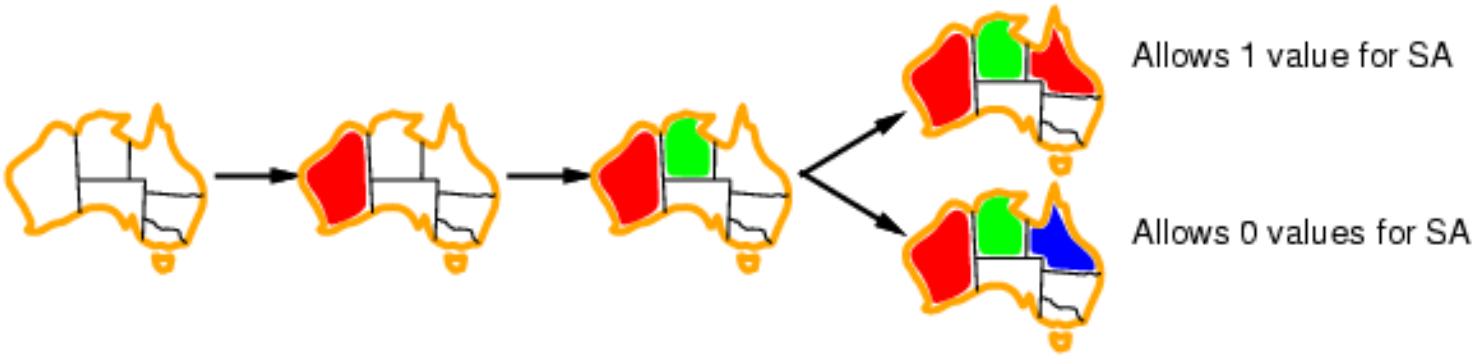



- Most constrain*ing* variable:
  - choose the variable with the most constraints on remaining variables (most edges in graph)



# In what order should its values be tried?
# → least constraining value heuristic


- *Given a variable*, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables




- Leaves maximal flexibility for a solution.
- Combining these heuristics makes 1000 queens feasible

# Rationale for MRV, DH, LCV

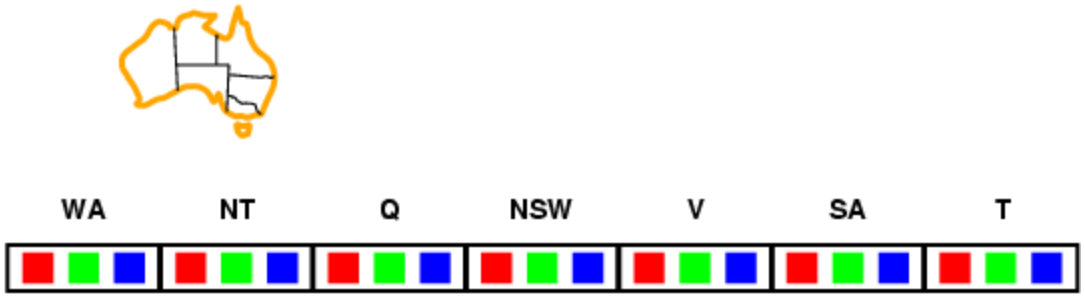
- In all cases we want to enter the most promising branch, but we also want to detect inevitable failure as soon as possible.

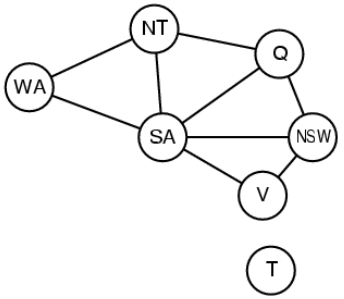
- MRV+DH: the variable that is most likely to cause failure in a branch is assigned first. The variable must be assigned at some point, so if it is doomed to fail, we'd better found out soon. E.g X1-X2-X3, values 0,1, neighbors cannot be the same.

- LCV: tries to avoid failure by assigning values that leave maximal flexibility for the remaining variables. We want our search to succeed as soon as possible, so given some ordering, we want to find the successful branch.

# Can we detect inevitable failure early? → forward checking

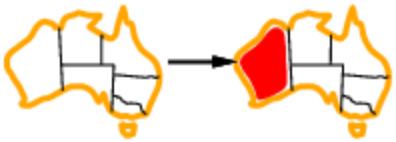
- Idea:
  - Keep track of remaining legal values for unassigned variables that are connected to current variable.
  - Terminate search when any variable has no legal values

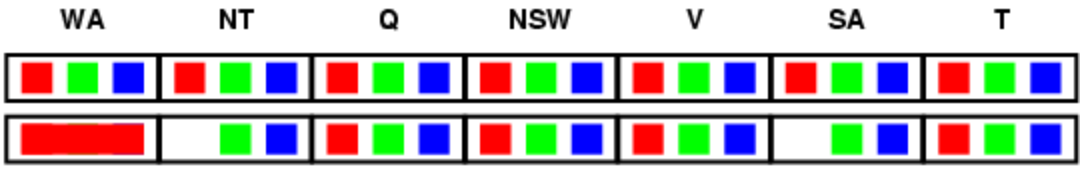


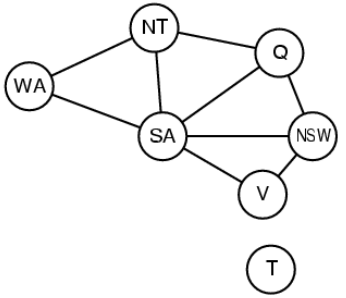



# Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

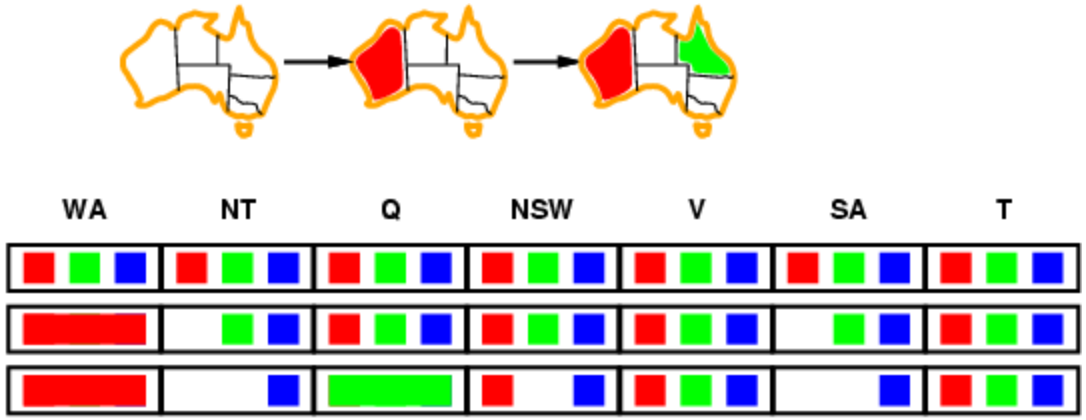


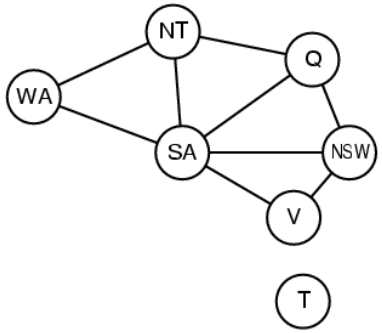


# Forward checking

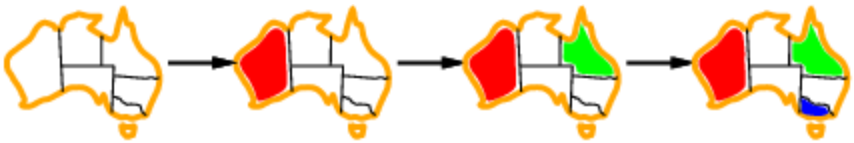
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

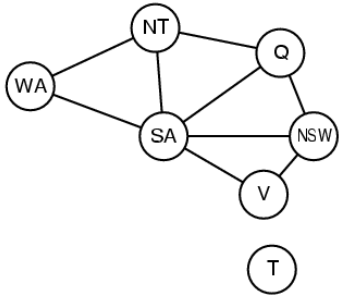


# Forward checking

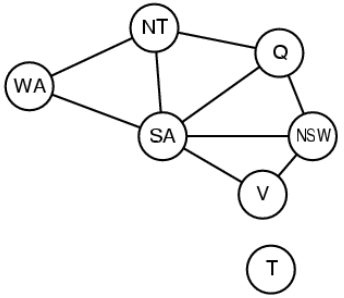
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



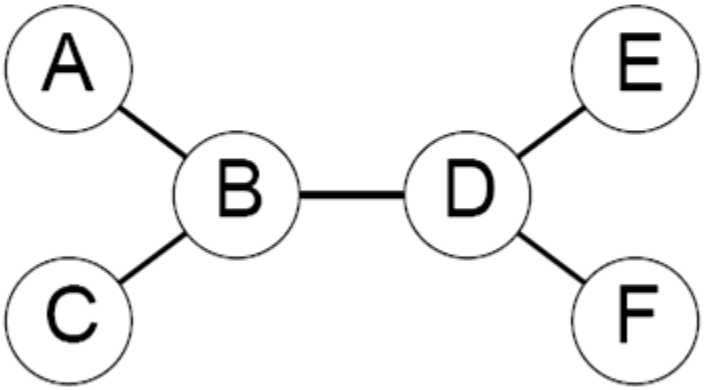
# Constraint propagation

- Forward checking only checks consistency between assigned and non-assigned states. How about constraints between two unassigned states?


- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

# Tree-structured CSPs

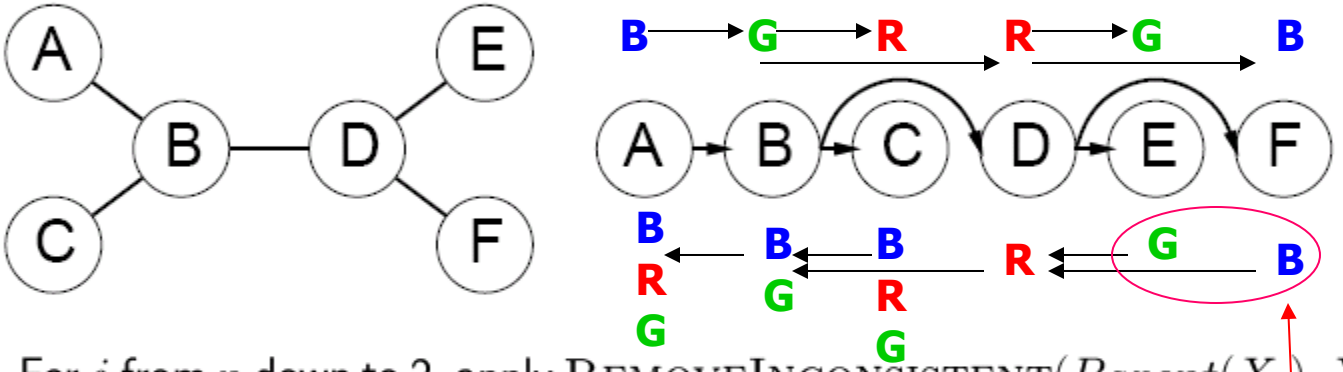



Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

# Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering




a priori constrained nodes

2. For $j$ from $n$ down to 2, apply REMOVEINCONSISTENT$(Parent(X_j), X_j)$

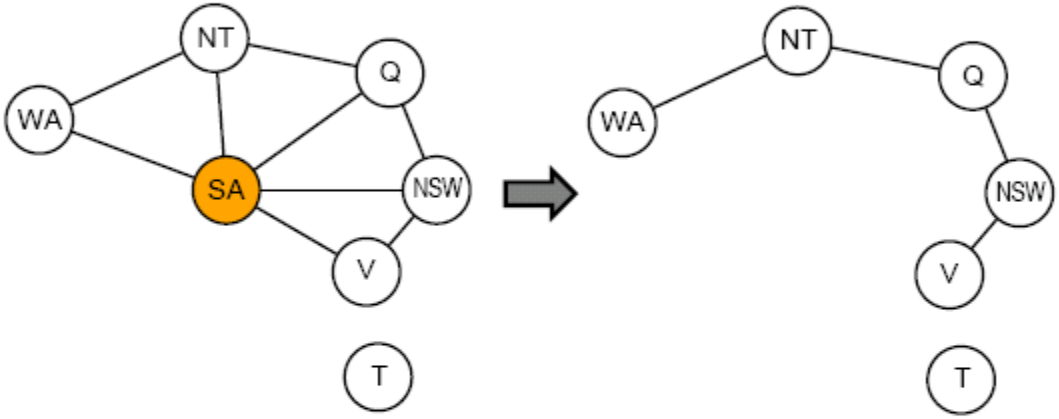
3. For $j$ from 1 to $n$, assign $X_j$ consistently with $Parent(X_j)$

Note: After the backward pass, there is guaranteed to be a legal choice for a child node for *any* of its leftover values.

This removes any inconsistent values from Parent(Xj), it applies arc-consistency moving backwards.

# Nearly tree-structured CSPs

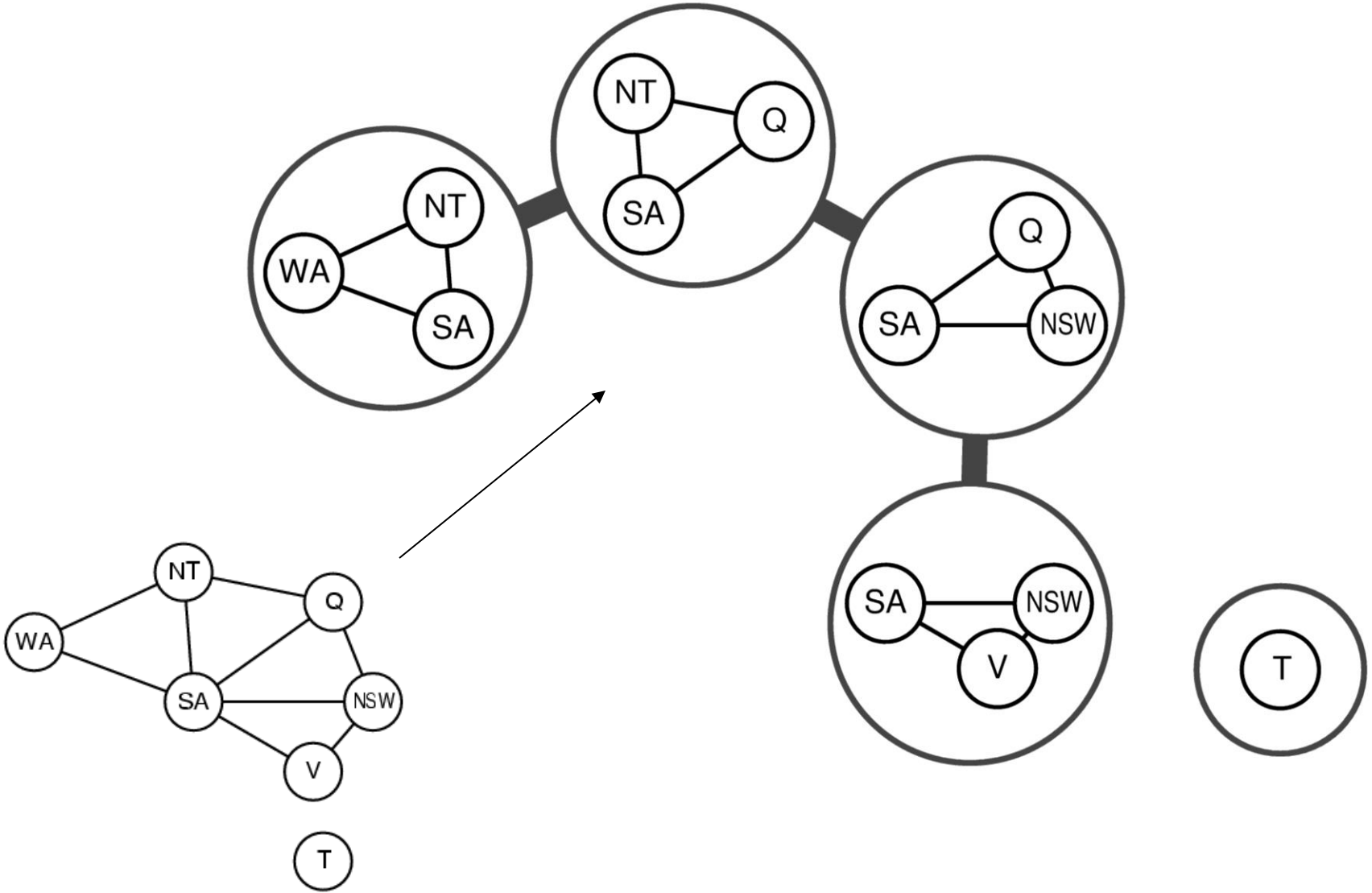
Conditioning: instantiate a variable, prune its neighbors' domains




Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c$ $\Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small $c$

# Junction Tree Decompositions

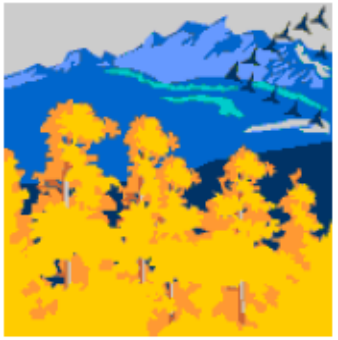


# Nature Inspired Algorithms for CSPs

Slides by Xiadong Li

## Swarm Intelligence

# Swarm Intelligence



**Swarm intelligence** (SI) is an artificial intelligence technique based around the study of collective behavior in decentralized, self-organized systems.

SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding and fish schooling (from *Wikipedia*).

# Swarm Intelligence

## Mind is social…

**Human intelligence results from social interaction**:
Evaluating, comparing, and imitating one another, learning from experience and emulating the successful behaviours of others, people are able to adapt to complex environments through the discovery of relatively optimal patterns of attitudes, beliefs, and behaviours. (Kennedy & Eberhart, 2001).

**Culture and cognition are inseparable consequences of human sociality**:
Culture emerges as individuals become more similar through mutual social learning. The sweep of culture moves individuals toward more adaptive patterns of thought and behaviour.

# Swarm Intelligence

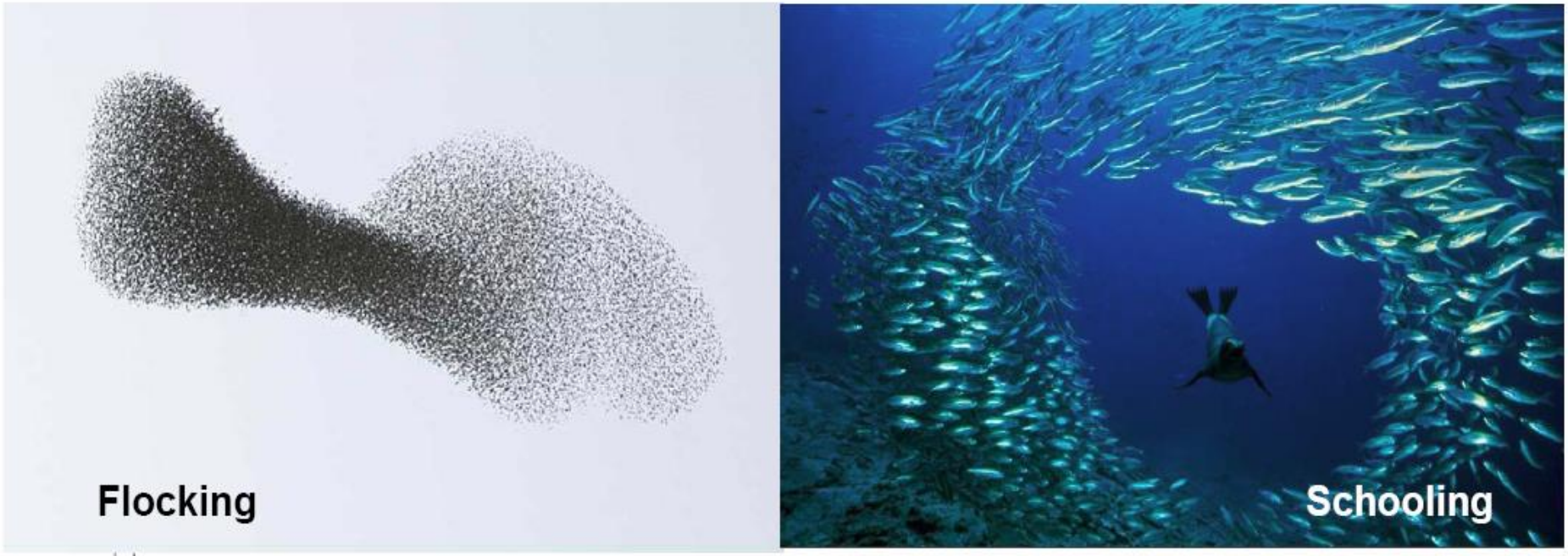
**To model human intelligence, we should model individuals in a social context, interacting with one another.**

# Emergent Collective Behavior

Some animal societies display coordinated and purposeful navigation of several individuals (from tens to thousands).

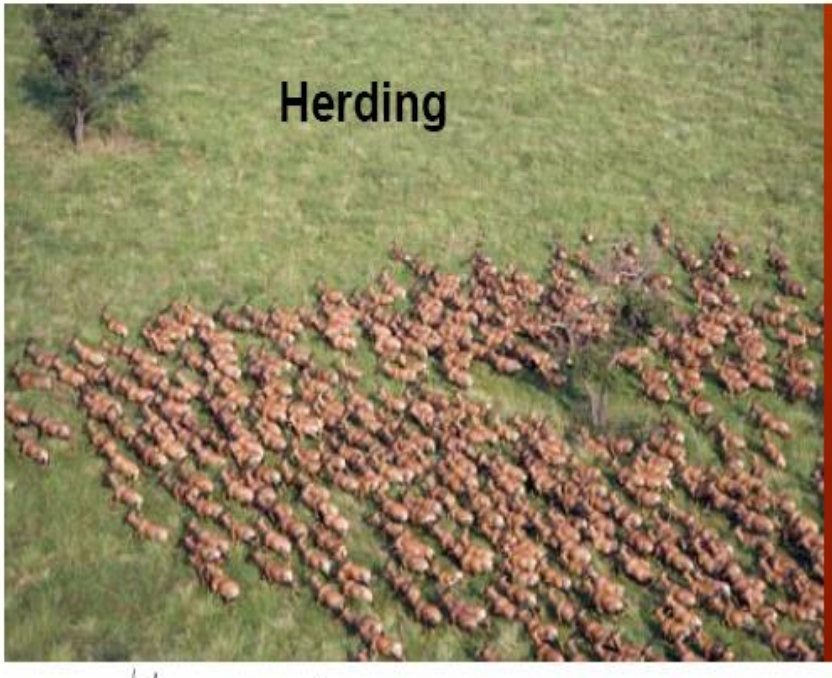
Each individual uses only local information about the presence of other individuals and of the environment.

There is no predefined group leader.



# Emergent Collective Behavior

In some cases there is a leader and more restrictive rules on relative motion, but individuals still use local information to decide how to move.





# *Swarm Intelligence*

Swarm Intelligence is the emergent collective intelligence of groups of simple individuals.

**Main principles:**

1) The swarm can solve complex problems that a single individual with simple abilities (computational or physical) could not solve.

2) The swarm is composed of several individuals, some of which may be lost or make mistake, but its performance is not affected.

3) Individuals in a swarm have local sensory information, perform simple actions, have little/no memory; they do not know the global state of the swarm or its goal.

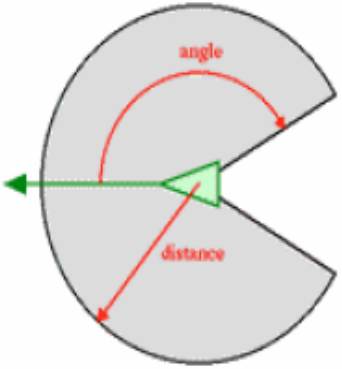
# Swarm Intelligence applications

- **Swarm-bots,** an EU project led by Marco Dorigo, aimed to study new approaches to the design and implementation of self-organizing and self-assembling artifacts (http://www.swarm-bots.org/).
- A 1992 paper by M. Anthony Lewis and George A. Bekey discusses the possibility of using swarm intelligence to control **nanobots** within the body for the purpose of killing cancer tumors.
- Artists are using swarm technology as a means of creating complex interactive environments.
  - Disney's ***The Lion King*** was the first movie to make use of swarm technology (the stampede of the bisons scene).
  - The movie "***Lord of the Rings***" has also made use of similar technology during battle scenes.

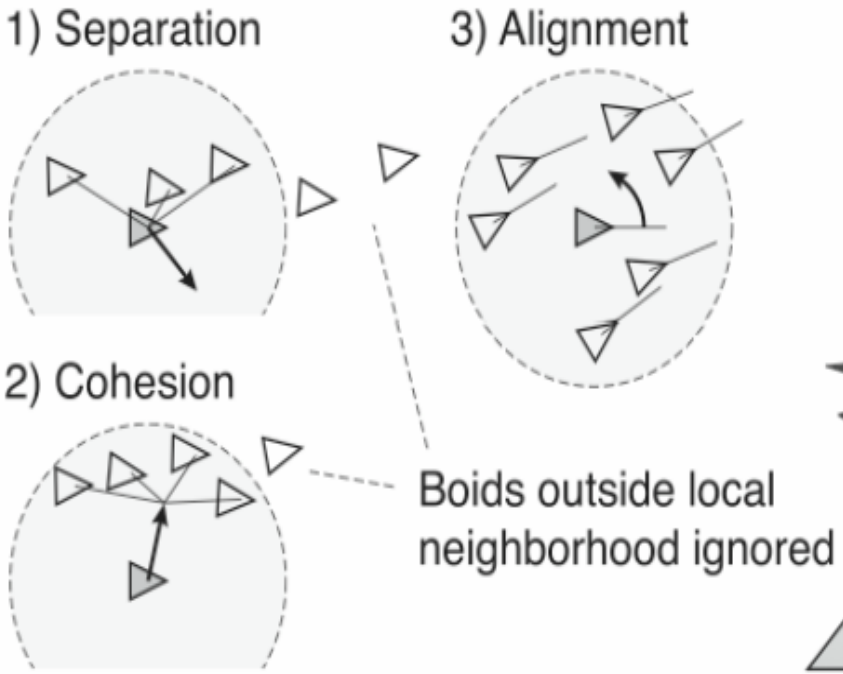
(Some examples from *Wikipedia*)

# *Reynolds Flocking (1987)*

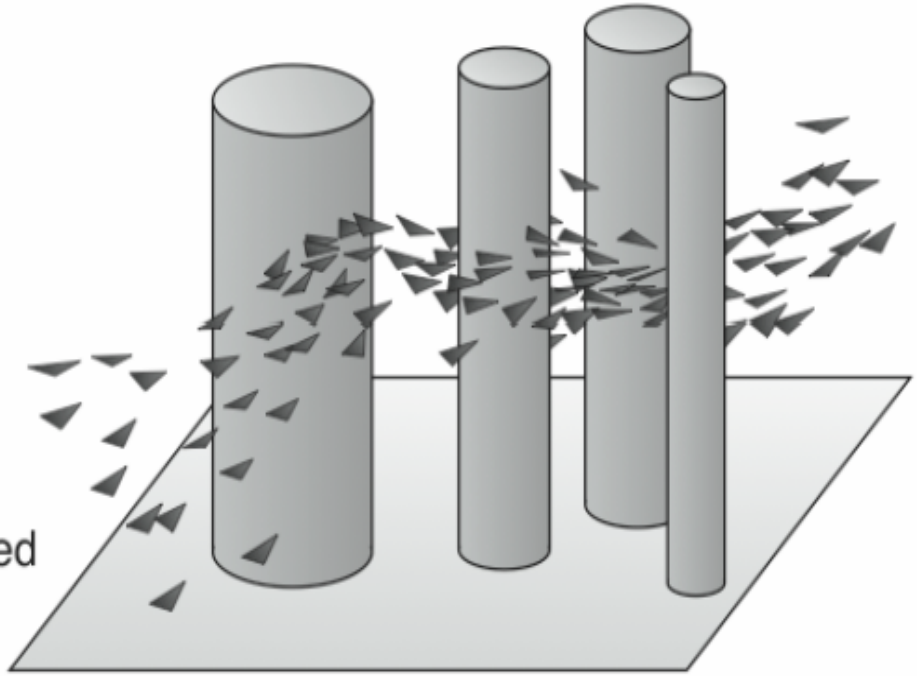
**Sensing**: Boid perceives angle and distance of neighboring boids




**Local rules**




**Emergent flocking behavior**



1. **Separation:** Boid maintains a given distance from other boids
2. **Cohesion:** Boid moves towards center of mass of neighboring boids
3. **Alignment:** Boid aligns its angle along those of neighboring boids

# Challenges of Swarm Intelligence

**Find individual behavioral rules that result in desired swarm behavior (reverse engineering).**

Fortunately, the challenge may be addressed because the behavioral rules are supposed to be relatively simple. Often rules are hand-designed, sometimes are evolved.

**Make sure the emergent behavior is stable.**

Dynamical systems theory may help to characterize and predict swarm behavior because a swarm can be described as a system of elements with negative and positive interactions that moves in space and time. However, non-linear interactions are still hard to model.

# Particle Swarm Optimization

Particle Swarm Optimization is an optimization algorithm inspired upon birds flocking to find the best food area.

**A caricature scenario**:
The flock wants to find the area with the highest concentration of food (insects). Birds do not know where that area is, but each bird can shout to their neighbors how many insects are at its location. Birds also remember their own location where they found the highest concentration of food so far.


The flock is most likely to succeed when birds combine **three strategies**:

1) **Brave**: keep flying in the same direction
2) **Conservative**: fly back towards its own best previous position
3) **Swarm**: move towards its best neighbor

# Particle Swarm Optimization

PSO has its roots in Artificial Life and social psychology, as well as engineering and computer science.

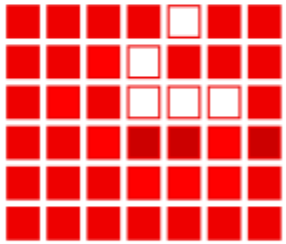
The particle swarms in some way are closely related to cellular automata (CA):

a) individual cell updates are done in parallel

b) each new cell value depends only on the old values of the cell and its neighbours, and

c) all cells are updated using the same rules (Rucker, 1999).

Blinker

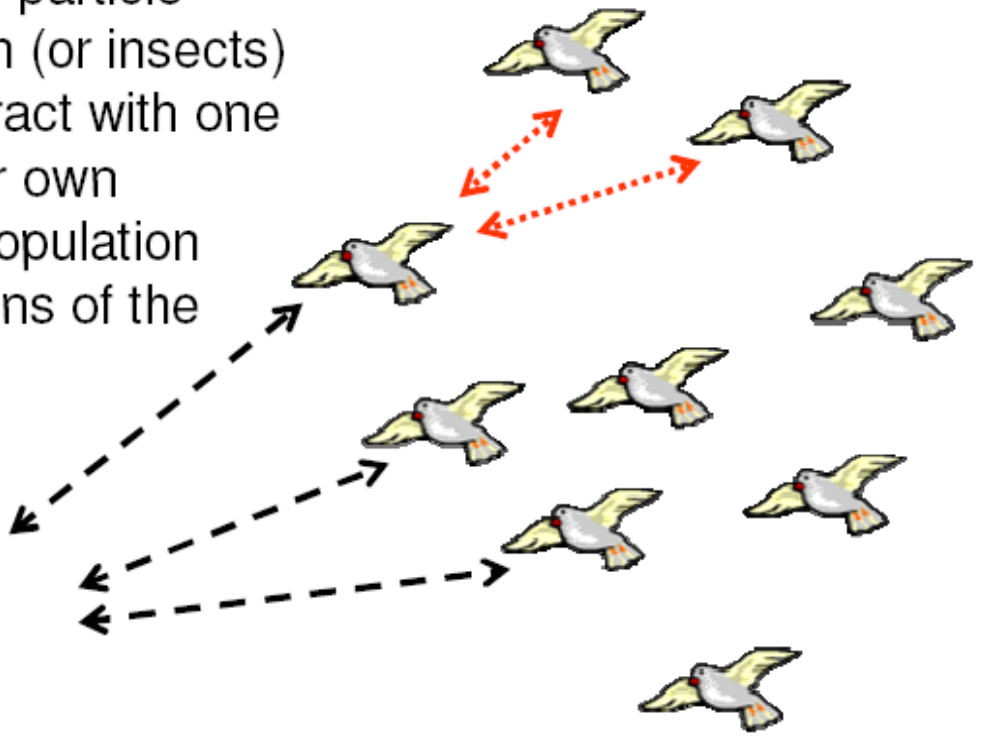


Glider

**Individuals in a particle swarm can be conceptualized as cells in a CA, whose states change in many dimensions simultaneously.**

# Particle Swarm Optimization

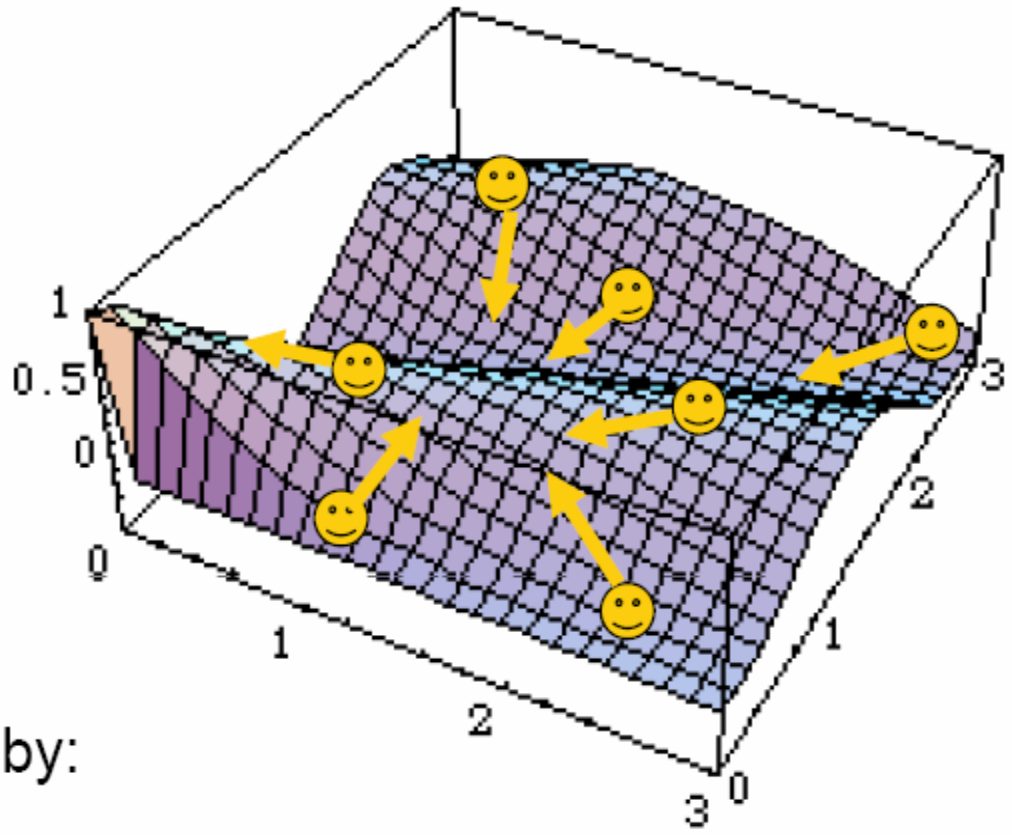
As described by the inventers James Kennedy and Russell Eberhart, “particle swarm algorithm imitates human (or insects) social behavior. Individuals interact with one another while learning from their own experience, and gradually the population members move into better regions of the problem space”.



Why named as “Particle”, not “points”? Both Kennedy and Eberhart felt that velocities and accelerations are more appropriately applied to particles.

# *From Birds to Particles*

The food concentration describes the search space of the optimization problem and the birds are the local solutions for that problem. They are called *particles* because they are very simple.




A particle **p** is described by:

**s[]** its position; e.g.: x, y

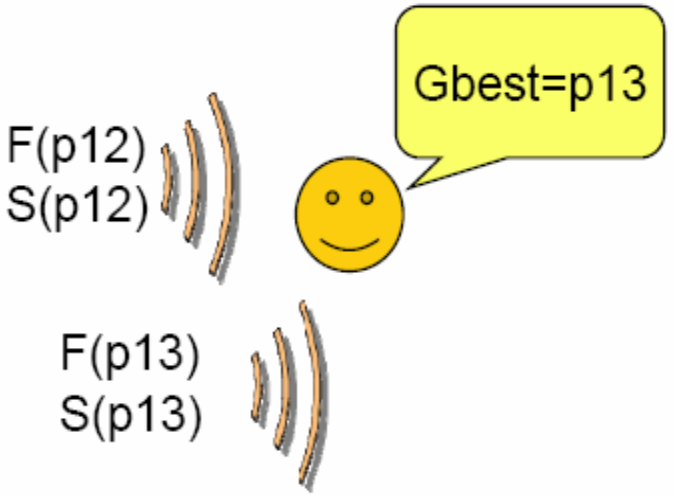
**v[]** its velocity; e.g. (for discrete case) angle and distance of next step

**f[]** its performance; e.g.: value of the function at its location

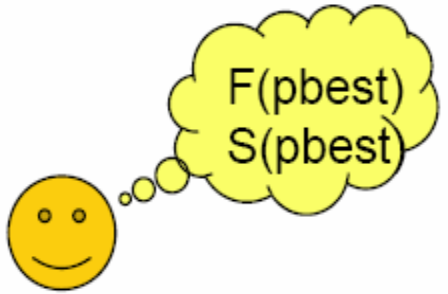
# Particle's perception

A particle perceives performances and positions of neighboring particles.

It can also tell which is the best particle among its neighbors (gbest)

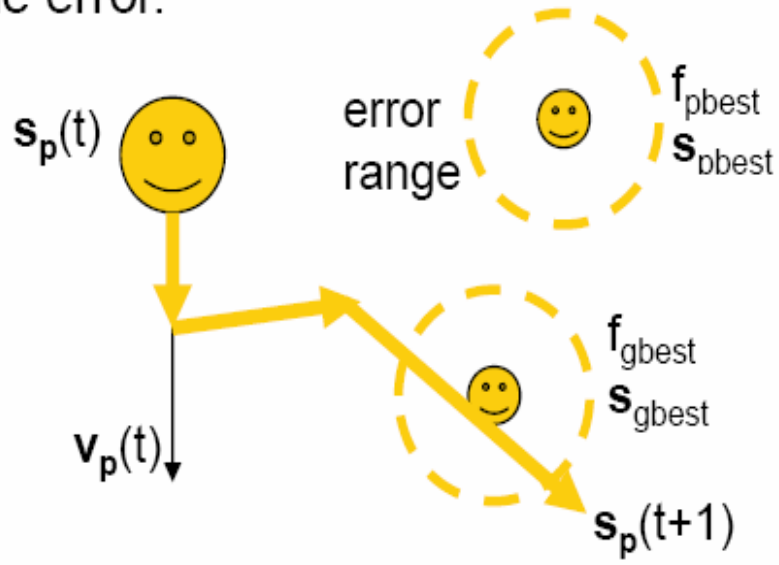



A particle remembers the position where it obtained the best performance so far (pbest)



# Particle's actions

A particle computes the next position by taking into account a fraction of its current velocity **v**, the direction to its previous best location pbest, and the direction to the location of the best neighbor gbest. The movement towards other particles has some error.




$$v_p(t+1) = a \times v_p(t) + b \times R \times (s_{pbest} - s_p(t)) + c \times R \times (s_{gbest} - s_p(t))$$

$$s_p(t+1) = s_p(t) + v_p(t+1)$$

where a, b, c are learning constants between 0 and 1
R is a random number between 0 and 1

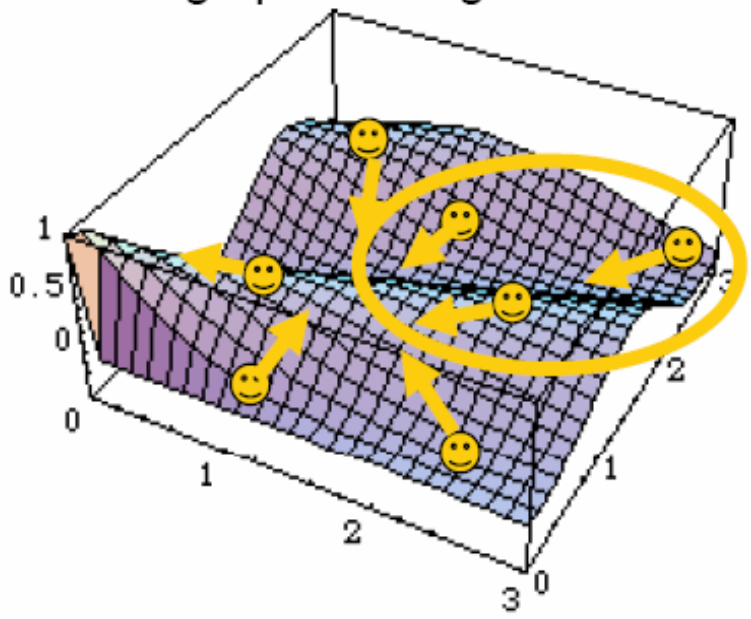
# Initialization

**Swarm size**: Typically 20 particles for problems with dimensionality 2 - 200

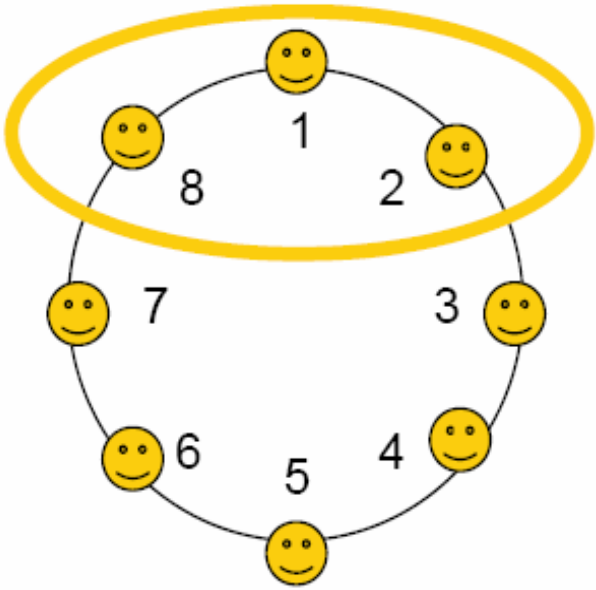
**Initial position** of each particle: Random

**Neighborhood topology**: Global, geographical or social (list based)

Geographical neighborhood




Social neighborhood




**Neighborhood size**: Typically 3 to 5

Set **max velocity** to $v_{max}$; if **v**(t+1) is larger, clip it to $v_{max}$

Iterate until best solution is found or no further improvement

# PSO vs. Artificial Evolution

As in Artificial Evolution, PSO works with a population and some random factor to update solutions.

Contrary to Artificial Evolution, there is no generation change, no genome, and no competition among the individuals (rather cooperation)

A major issue in PSO is to transform the parameters of the problem to be solved so that it can be encoded and searched by particles

The best applications found so far include the large class of Traveling Salesman Problems and the optimization of neural network weights.

**Reference**: Kennedy and Eberhart (2001) Swarm Intelligence. Morgan Kauffman

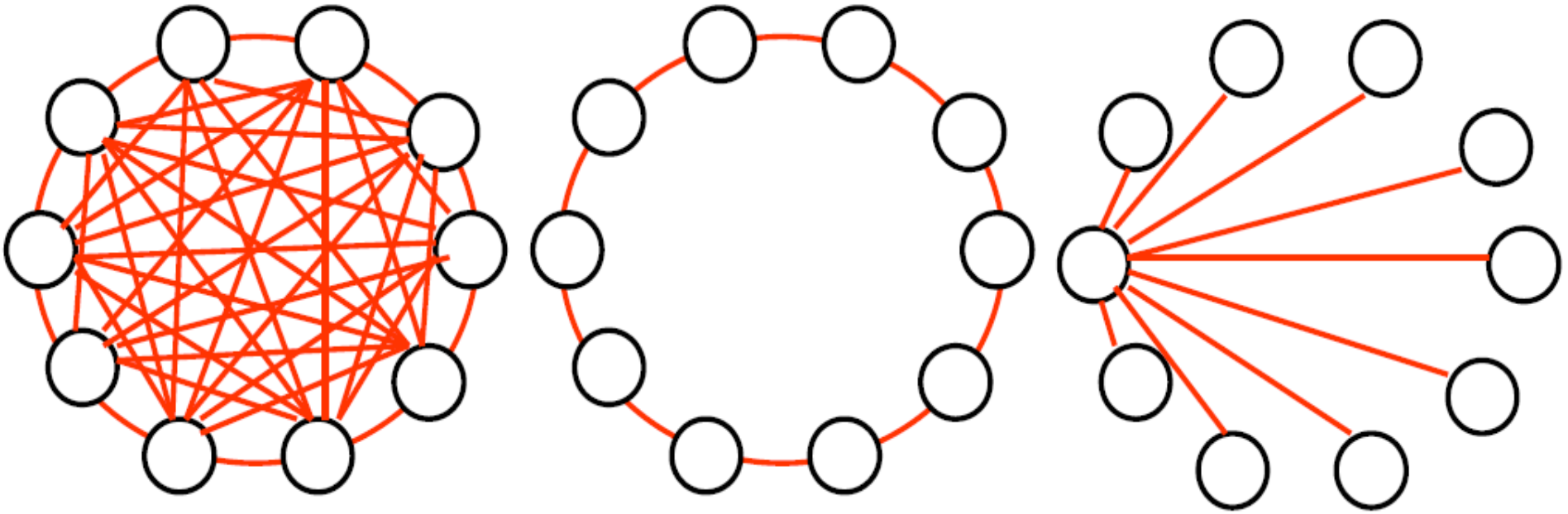
# PSO applications

Problems with continuous, discrete, or mixed search space, with multiple local minima.

- Evolving neural networks:
  - Human tumor analysis;
  - Computer numerically controlled milling optimization;
  - Battery pack state-of-charge estimation;
  - Real-time training of neural networks (Diabetes among Pima Indians);
  - Servomechanism (time series prediction optimizing a neural network);
- Reactive power and voltage control;
- Ingredient mix optimization;
- Pressure vessel (design a container of compressed air, with many constraints);
- Compression spring (cylindrical compression spring with certain mechanical characteristics);
- Moving Peaks (multiple peaks dynamic environment); and more

PSO can be tailor-designed to deal with specific real-world problems.

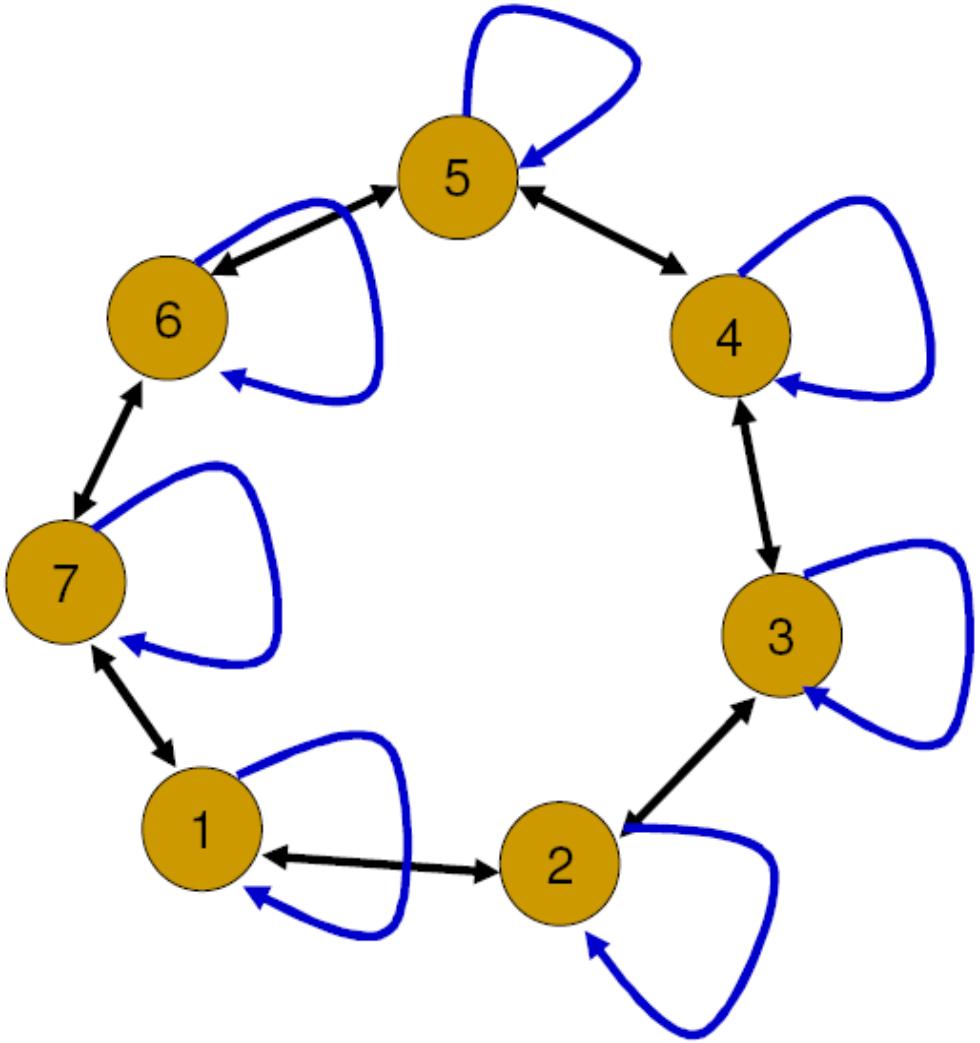
# Communication topologies (1)



Two most common models:

- **gbest**: each particle is influenced by the best found from the entire swarm.
- **lbest**: each particle is influenced only by particles in local neighbourhood.

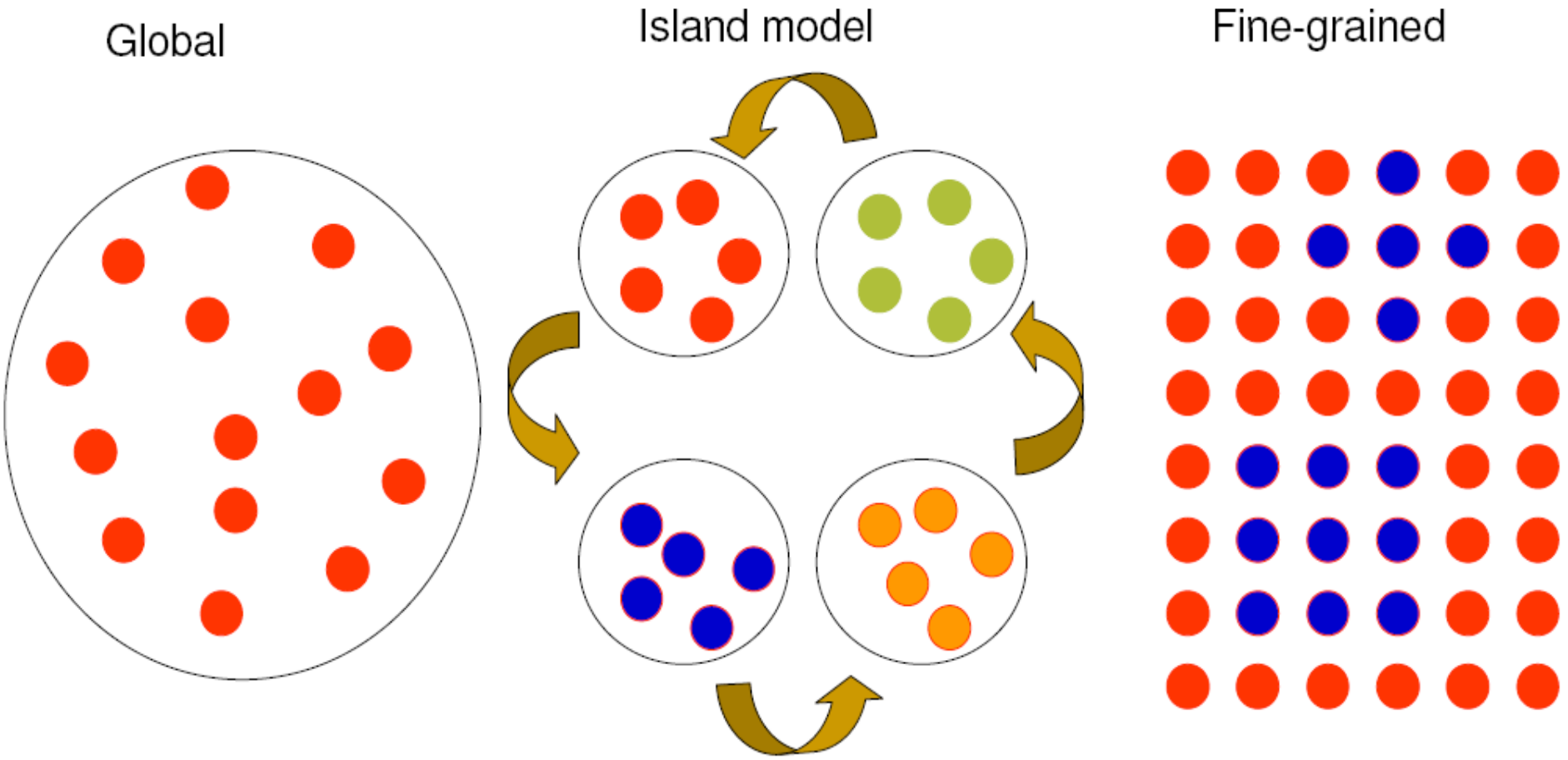
# Communication topologies (2)




Graph of influence of a swarm of 7 particles. For each arc, the particle origin influence (informs) the end particle (Clerc, 2006)

This graph of influence can be also expanded to include previous best positions.

# Communication topologies (3)



# Communication topologies (4)

## Which one to use?

Balance between exploration and exploitation…

**gbest** model propagate information the fastest in the population; while the **lbest** model using a ring structure the slowest. For complex multimodal functions, propagating information the fastest might not be desirable. However, if this is too slow, then it might incur higher computational cost.

Mendes and Kennedy (2002) found that von Neumann topology (north, south, east and west, of each particle placed on a 2 dimensional lattice) seems to be an overall winner among many different communication topologies.

# Multimodal problems

# Optimization in a dynamic environment

Many real-world optimization problems are dynamic and require optimization algorithms capable of adapting to the changing optima over time.

E.g., Traffic conditions in a city change dynamically and continuously. What might be regarded as an optimal route at one time might not be optimal in the next minute.

**In contrast to optimization towards a static optimum, in a dynamic environment the goal is to track as closely as possible the dynamically changing optima.**

# Why PSO?

- With a population of candidate solutions, a PSO algorithm can maintain useful information about characteristics of the environment.
- PSO, as characterized by its fast convergence behaviour, has an in-built ability to adapt to a changing environment.
- Some early works on PSO have shown that PSO is effective for locating and tracking optima in both static and dynamic environments.

**Two major issues** must be resolved when dealing with dynamic problems:

- How to **detect** that a change in the environment has actually occurred?
- How to **respond** appropriately to the change so that the optima can still be tracked?

# Set the scope

Many complex scenarios are possible:

- Small and continuous changes;
- Large, random and infrequent changes;
- Large and frequent changes.

**Assumption**:
Here we assume that changes are only slight in a dynamic environment. It would be beneficial to use knowledge about the old environment to help search in the new environment.

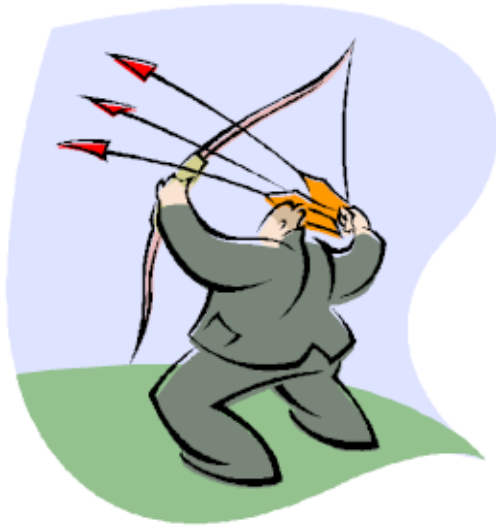
- Speciation-based PSO is able to identify peaks and converge onto these peaks in parallel and adaptively.
- It can be further enhanced by other techniques (eg., quantum swarms) to better track changing optima.

# Multiobjective optimization

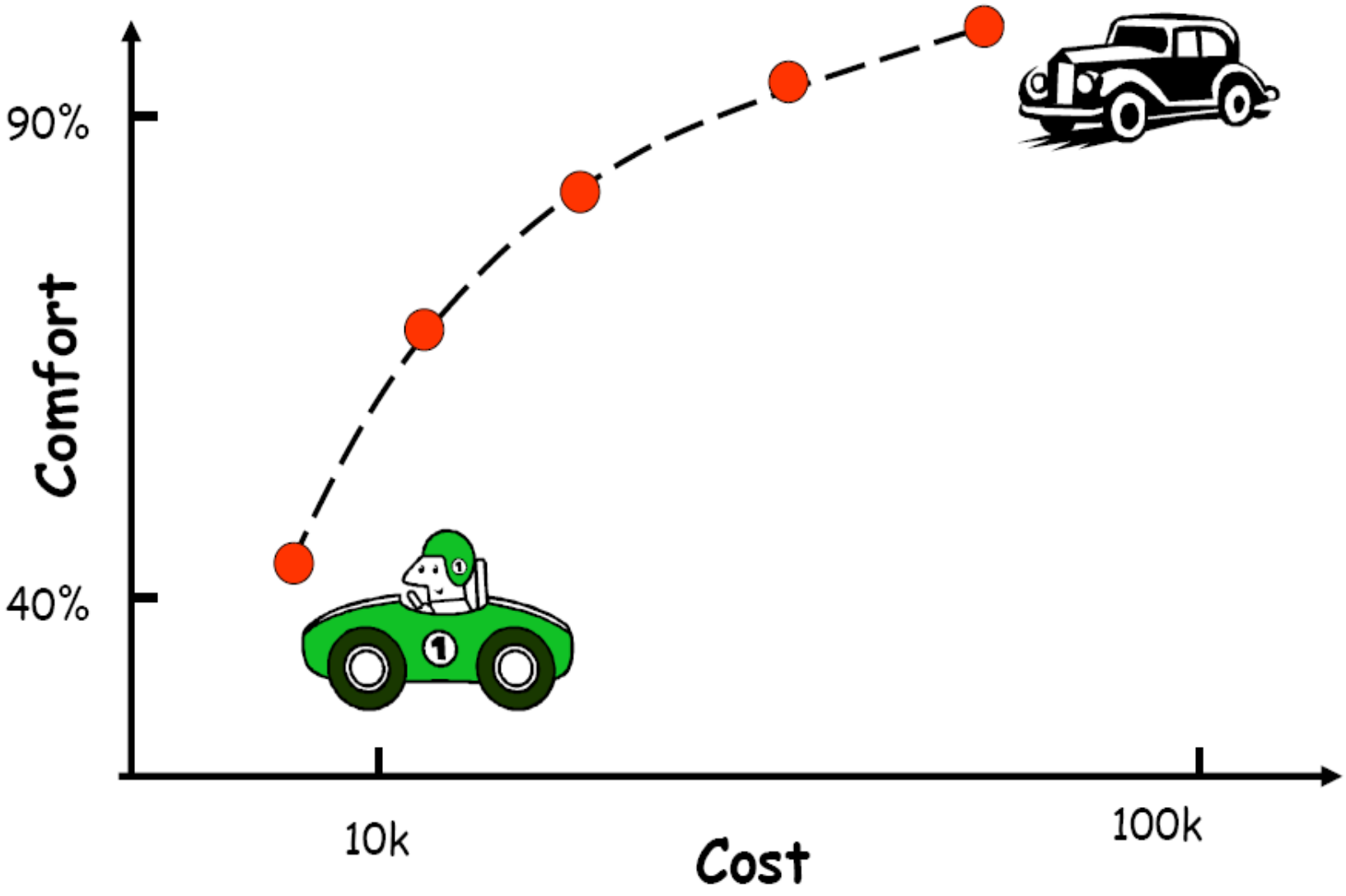
***"The great decisions of human life have as a rule far more to do with the instincts and other mysterious unconscious factors than with conscious will and well-meaning reasonableness. The shoe that fits one person pinches another; there is no recipe for living that suits all cases. Each of us carries his own life-form - an indeterminable form which cannot be superseded by any other."***

Carl Gustav Jung, Modern Man in Search of a Soul, 1933, p. 69

Many real-world problems involve multiple conflicting objectives, which need to be optimized simultaneously. The task is to find the best possible solutions which still satisfy all objectives and constraints. This type of problems is known as multiobjective optimization problems.



# Multiobjective optimization



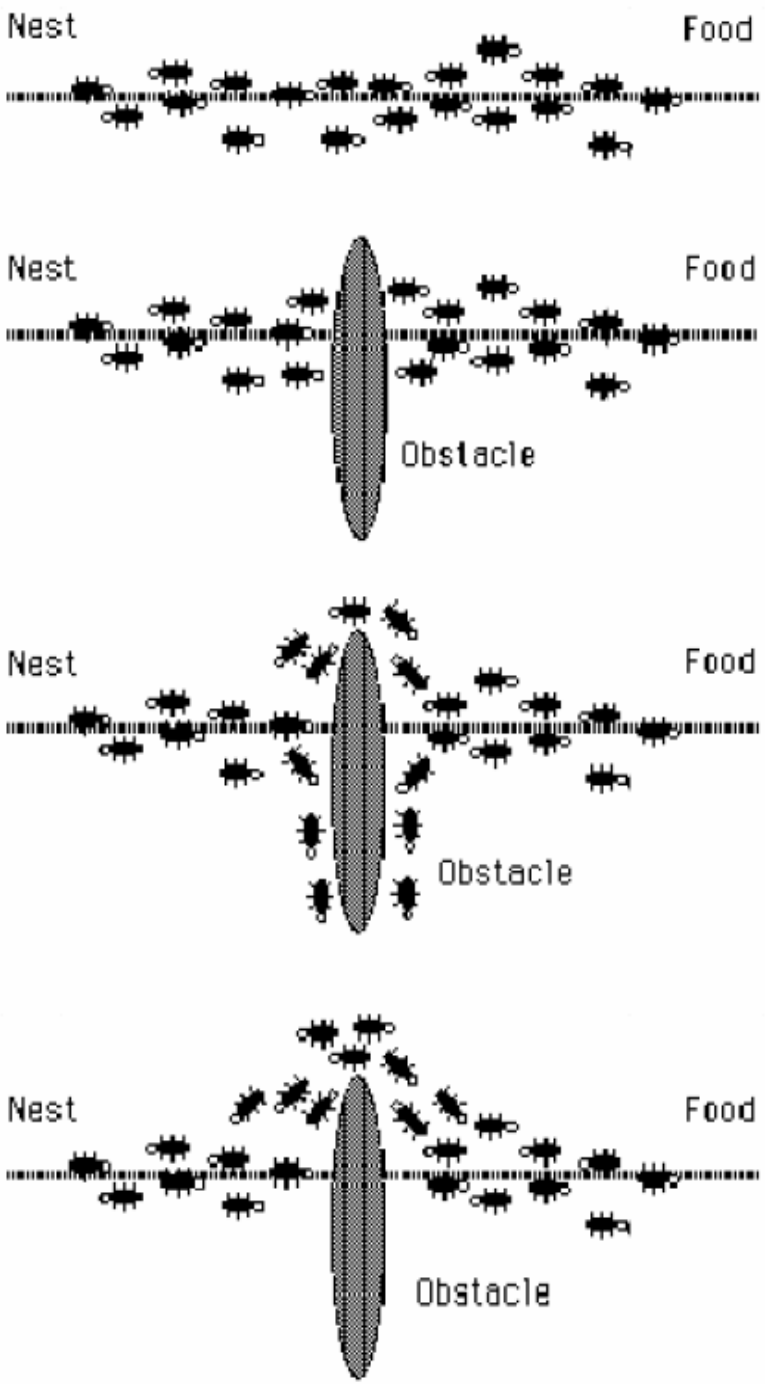
# Ant trails

# *Stigmergy*

The term indicates communication among individuals through modification of the environment.

For example, some ants leave a chemical (pheromone) trail behind to trace the path. *The chemical decays over time.*

This allows other ants to find the path between the food and the nest. It also allows ants to find the shortest path among alternative paths.

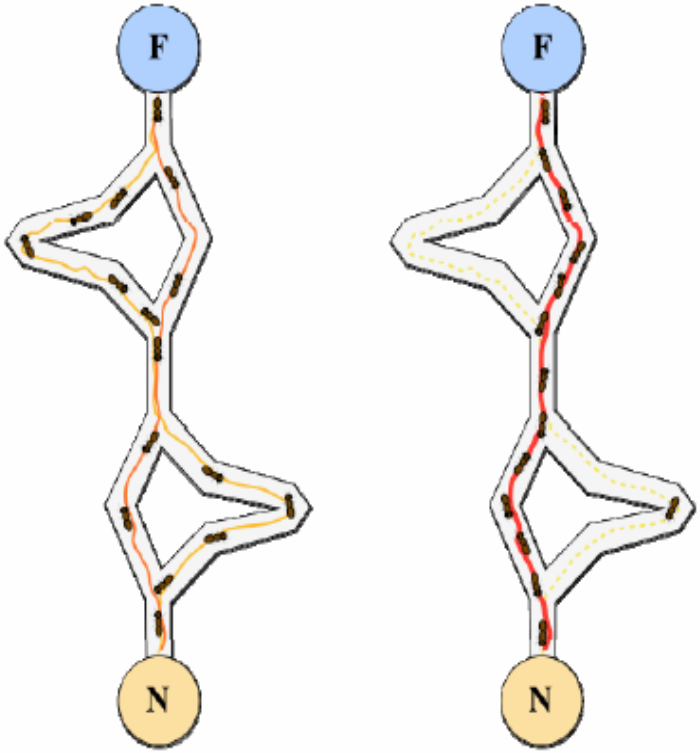


# *Finding the Shortest Path*

1) As they move, ants deposit pheromone
2) Pheromone decays in time
3) Ants follow path with highest pheromone concentration
4) Without pheromone, equal probability of choosing short or long path

Shorter path allows higher number of passages and therefore pheromone level will be higher on shorter path.

Ants will increasingly tend to choose shorter path.



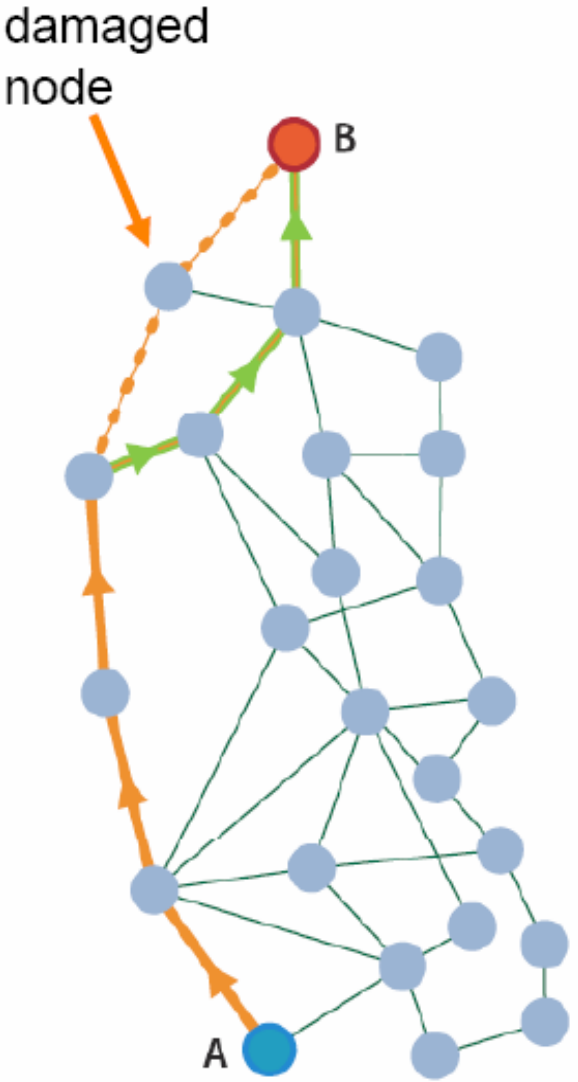

Goss et al. 1989, Deneubourg et al. 1990

# Ant Colony Optimization

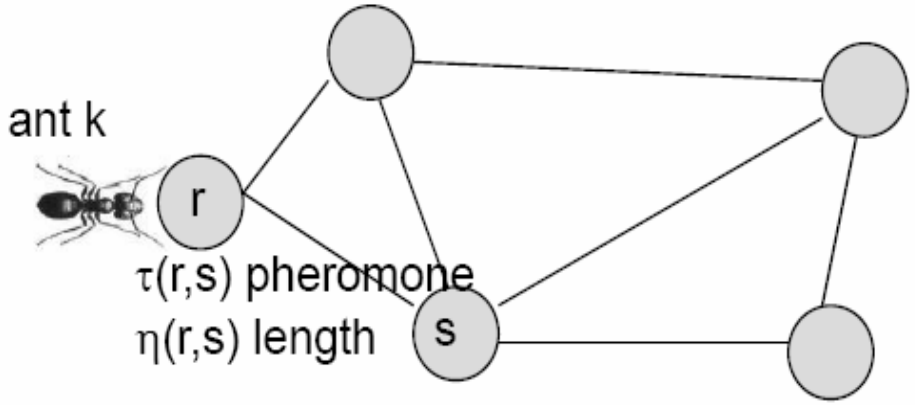
Ant Colony Optimization is an algorithm developed by Dorigo et al. in 1994 inspired upon stigmergic communication to find the shortest path in a network.

Typical examples are telephone, internet, and any problem that can be described as Travel Salesman Problem. Used/adopted by British Telecom, MCI Worldcom, Barilla, etc.

Advantage of algorithm is that, as ants do, it allows dynamic rerouting through shortest path if one node is broken. Most other algorithms instead assume that the network is static.



# Ant Colony Optimization



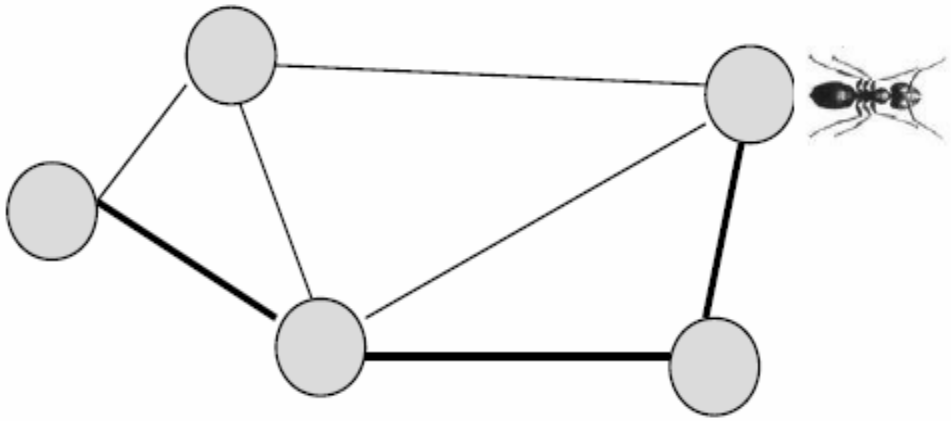

Each ant generates a complete tour of nodes using probabilistic transition rule encouraging choice of edge with high pheromone and short distance

Pheromone level on each edge is updated by considering evaporation and deposit by each ant

Pheromone levels only of edges traveled by best ant are increased in inverse proportion to length of path.

Result is that edges that belong to short tours receive greater amount of pheromone

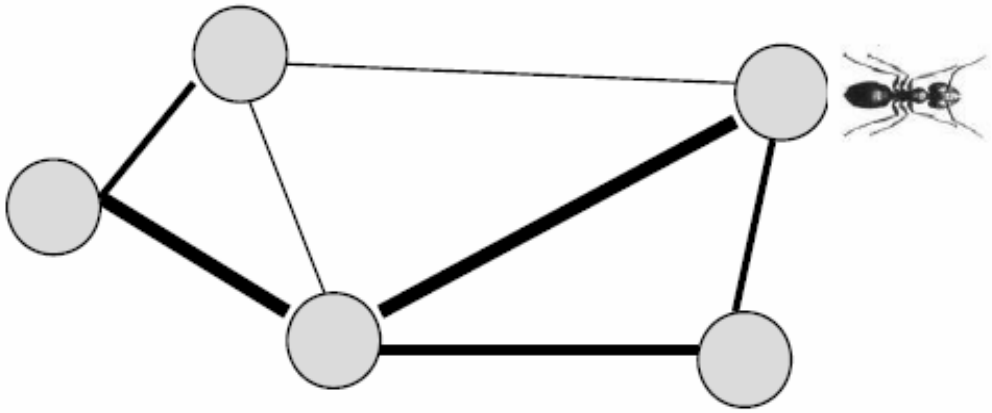
# Pheromone Level Update: Local



$$\tau(r,s) \leftarrow (1-\rho)\cdot \tau(r,s) + \rho \cdot \tau 0$$

The pheromone level of each edge visited by an ant is decreased by a fraction (1- $\rho$) of its current level and increased by a fraction $\rho$ of the initial level $\tau 0$.

# Pheromone Level Update: Global



$$\tau(r,s) \leftarrow (1-\rho)\cdot \tau(r,s) + \rho \cdot L^{-1}$$

When all ants have completed their tours, the length L of the shortest tour is found and the pheromone levels of only the edges of this shortest path are updated in inverse proportion to the path length.

# Initialization

Use approximately 100 ants

Distribute them on random nodes

Initial pheromone level is equal for all edges and inversely proportional to number of nodes times estimated length of optimal path

Initial pheromone level $\tau 0 = \left(n \cdot L_{nn}\right)^{-1}$

Importance of length over pheromone $\beta = 2$

Exploration threshold $q0 = 0.9$

Pheromone update rate $\rho = 0.1$

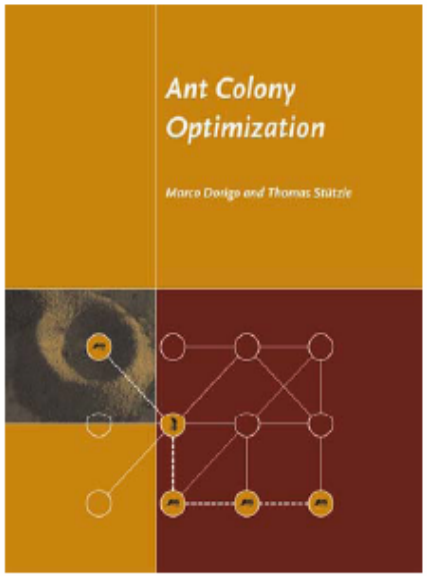
# ACO Performance

Finds best solution on “small” problems (up to 30 cities)

Finds good solutions on large problems compared to other techniques

Finds best solution on large problems when coupled with other search techniques

Can operate on dynamic problems (e.g., node malfunctioning) that require fast rerouting




Dorigo and Stuetzle, 2005, MIT Press

# The End!