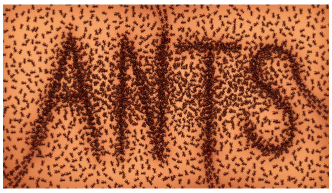
Swarm Intelligence (Vienna 2022)

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)



Outline

1. Swarm intelligence
2. Different metaheuristics:
   - Ant Colony Optimization (ACO)
   - Particle Swarm Optimization (PSO)
   - Artificial Bee Colony (ABC) algorithm
3. Applications: examples from our work

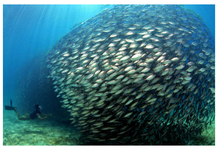
© Alex Wild

(`http://www.myrmecos.net`)

Swarm intelligence

In a nutshell

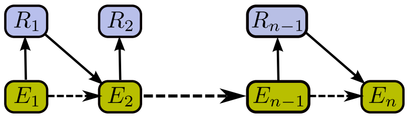
AI discipline whose goal is designing intelligent multi-agent systems by taking inspiration from the collective behaviour of animal societies such as ant colonies, flocks of birds, or fish schools.



Swarm intelligence (SI)

Characteristics of SI systems

- Consist of a set of simple entities
- Distributedness: No global control
- Self-organization through:
  - **Direct communication:** visual or chemical contact
  - **Indirect communication: Stigmergy** (Grassé, 1959)



Result: Complex tasks/behaviors can be accomplished/exhibited in cooperation

Example: flocks of birds

Bird flocking: a model

Definition of flocking

The collective motion of a large number of self-propelled entities

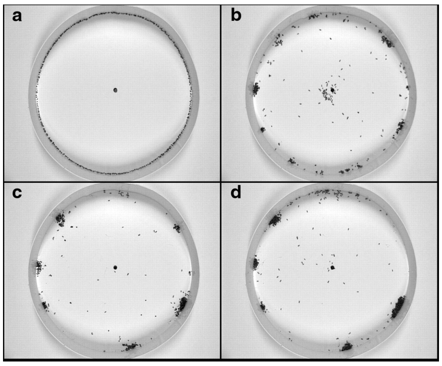
Note

- Commonly used as a demonstration of emergence and self-organization
- Modelled/simulated for the first time by Craig Reynolds (Boids, 1986)

Model: basic rules

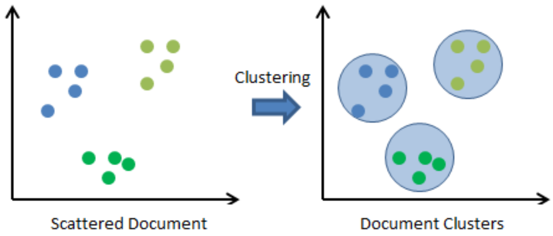
1. Separation: avoid crowding neighbours (short range repulsion)
2. Alignment: steer towards average heading of neighbours
3. Cohesion: steer towards average position of neighbours (long range attraction)

Example: cemetery formation



Cemetery formation: utilization

Used for: object clustering



Ex.: division of labour / task allocation

Leaf cutter ants

Army ants

Division of labour: Why is it necessary?

- **Problem:** in any colony (ants, bees, etc) are a number of tasks to fulfill
- **Examples:** brood tending, foraging for resources, maintaining the nest
- **Requires:** dynamic allocation of individuals to tasks
- **Depends on:** state of the environment, needs of the colony
- **Requires:** global assessment of the colonies current state

However: Individuals are unable (as individuals) to make a global assessment

Solution: response threshold models

Response threshold models (1)

Assume that:

- There are m tasks to fulfill
- The colony consists of n individuals
- Each individual i has a response threshold δ_{ij} for each task j
- Let $s_j \geq 0$ be the stimulus of task j
- An individual i engages in task j with probability

$$p_{ij} = \frac{s_j^2}{s_j^2 + \delta_{ij}^2}$$

This means:

- If $s_j << \delta_{ij}$: p_{ij} is close to 0
- If $s_j >> \delta_{ij}$: p_{ij} is close to 1

Response threshold models (2)

This means (continued ...):

- Si $s_j = \delta_{ij}$: $p_{ij} = 0.5$
- When s_j is rather low, only individuals i with a low δ_{ij} respond

Additional feature: response thresholds are dynamic

- Let Δ_t be a unit time duration.
- Let $x_{ij}\Delta_t$ be the fraction of time spend by i on task j within Δ_t
- Then: $(1 - x_{ij})\Delta_t$ is the time spent by i on other tasks

Response threshold update:

$$\delta_{ij} \rightarrow \delta_{ij} - \xi x_{ij}\Delta t + \rho(1 - x_{ij})\Delta t$$

Response threshold models (3)

Where:

- ξ is a reinforcement coefficient
- ρ is a forgetting coefficient

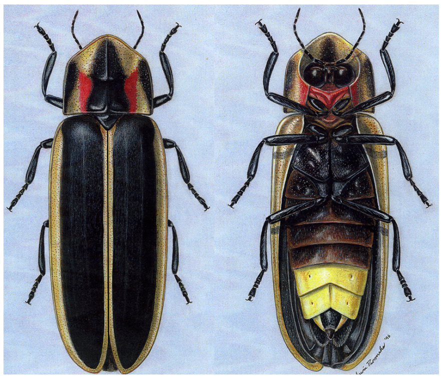
Effects:

- The more an individual engages in a task j, the lower becomes its threshold
- The less an individual engages in a task j, the higher becomes its threshold

Utilization of these models:

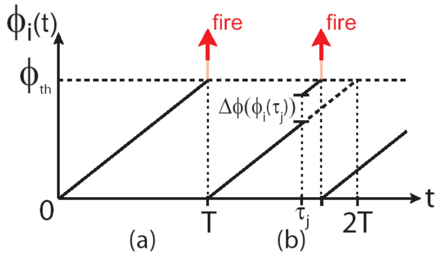
- Dynamic problems
- Swarm robotics

Example: self-synchronization of fireflies



By curtesy of `www.learner.org`

Pulse-coupled oscillator models



© A. Tyrrell and G. Auer

Utilization of these models:

- Self-synchronization of clocks in sensor networks
- Combinatorial optimization

Example: fish schools

Utilization: combinatorial optimization

Example: nest construction

Termite mound

Ant hill

Automated construction

Utilization: swarm robotics

Example *Waggle dance* (bees)

Utilization: combinatorial optimization

Example: ants, shortest-path finding

Communication strategies:

- Direct communication: for example, recruitment
- Indirect communication: via chemical pheromone trails

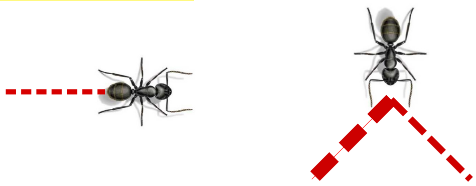
© Alex Wild (http://www.myrmecos.net)

© Christian Blum

Example: ants, shortest-path finding

Communication strategies:

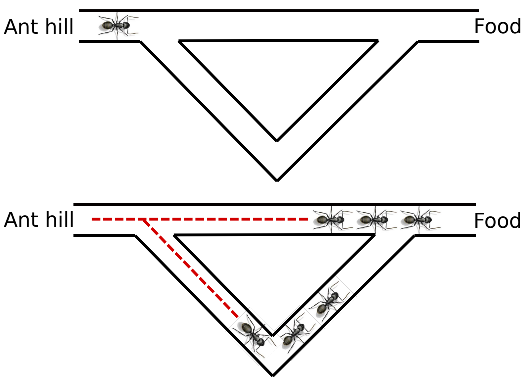
- Direct communication: for example, recruitment
- Indirect communication: via chemical pheromone trails



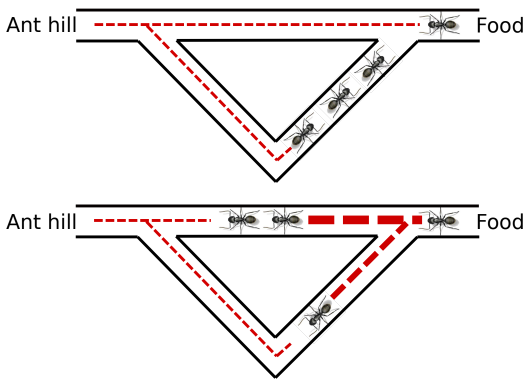
Foraging behaviour of ants

© Alex Wild (http://www.myrmecos.net)

Ants: *Double-bridge* experiment



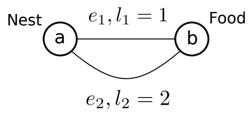
Ants: *Double-bridge* experiment



Double-bridge experiment: real

Double-bridge experiment: simulation

Technical simulation:



1. Definition of the artificial pheromone parameters:

   $\mathcal{T}_1$ for e_1 and $\mathcal{T}_2$ for e_2

2. Initialization of the pheromone values:

   $\tau_1 = \tau_2 = c > 0$

Double-Bridge simulation: algorithm

Iterate the following steps:

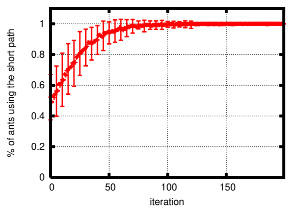
1. Place n_a ants in node a.
2. Each of the n_a ants traverses from a to b, either
   - via e_1 with probability $\mathbf{p}_1 = \frac{\tau_1}{\tau_1+\tau_2}$,
   - or via e_2 with probability $\mathbf{p}_2 = 1 - \mathbf{p}_1$.
3. Evaporate the artificial pheromone: $i = 1, 2$

   $\tau_i \leftarrow (1 - \rho)\tau_i \ , \ \rho \in (0, 1]$

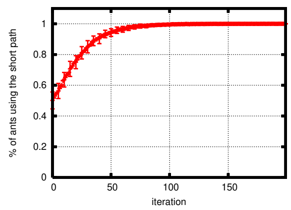
4. Each ant leaves pheromone on its traversed edge e_i:

   $$\tau_i \leftarrow \tau_i + \frac{1}{l_i}$$

Double-bridge simulation: result



10 ants



100 ants

Observation: Optimization capability is due to cooperation

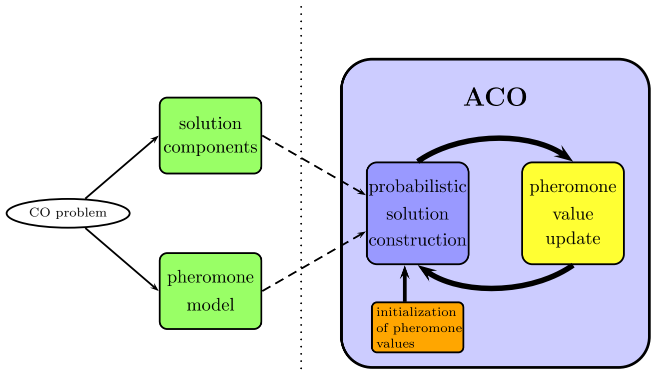
Differences between model and reality

	Real ants	**Simulation**
Movement of ants	asynchronous	synchronized
Pheromone laying	while moving	after the trip
Solution evaluation	implicitly	explicit quality measure

Problem: In combinatorial optimization we want to find good solutions (instead of enumerating all feasible solutions)

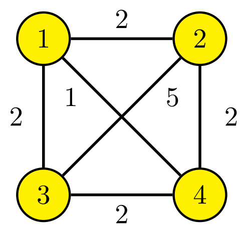
Questions?

Metaheuristic: Ant Colony Optimization



Example application: TSP

In the Travelling Salesman Problem (TSP) we are given a complete graph $G = (V, E)$ with weights/distances on the edges.



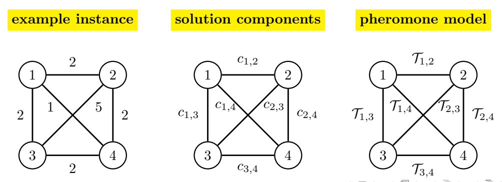
Objective: Among all Hamiltonian cycles in G, for which the sum of the weights on its edges is minimal.

TSP example: problem definition

TSP expressed in terms of e $\mathcal{P} = (\mathcal{S}, D, f)$:

- $\mathcal{S}$ is composed of all subsets of E with exactly $n = |V|$ edges
- $D(S \in \mathcal{S}) =$ valid solution, if the edges in S form a Hamiltonian cycle.
- The objective function $f(S)$ is defined as the sum of the weights of those edges that form part of S.

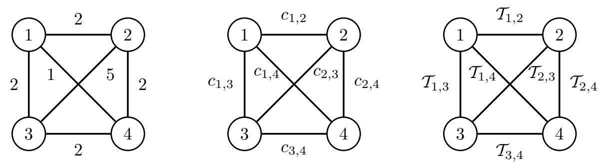
Example:



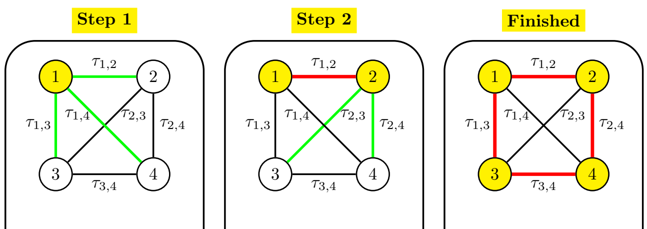
TSP example: pheromone model

Preliminary step: Definition of the ...

- solution components
- pheromone model



TSP example: solution construction



Step 1: $\mathbf{p}(c_{i,j}) = \dfrac{\tau_{i,j}}{\tau_{1,2} + \tau_{1,3} + \tau_{1,4}}$

Step 2: $\mathbf{p}(c_{i,j}) = \dfrac{\tau_{i,j}}{\tau_{2,3} + \tau_{2,4}}$

TSP example: pheromone update

Pheromone update: *Ant System (AS)* [Dorigo et al., 1996]

pheromone evaporation

$$\tau_{i,j} \leftarrow (1 - \rho) \cdot \tau_{i,j}$$

reinforcement

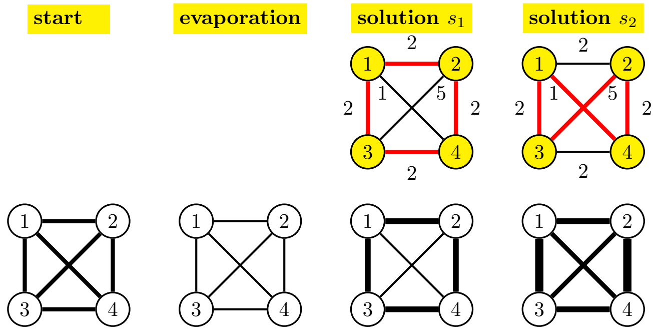
$$\tau_{i,j} \leftarrow \tau_{i,j} + \rho \cdot \sum_{\{S \in \mathcal{S}_{iter} | c_{i,j} \in S\}} F(S)$$

where

- $\rho \in (0, 1]$ is the evaporation rate
- $\mathcal{S}_{iter}$ is the set of solutions generated in the current iteration
- $F()$ is the quality function. For minimization we might use $F(\cdot) = \frac{1}{f(\cdot)}$

TSP example: pheromone update

Graphical illustration:



Questions?

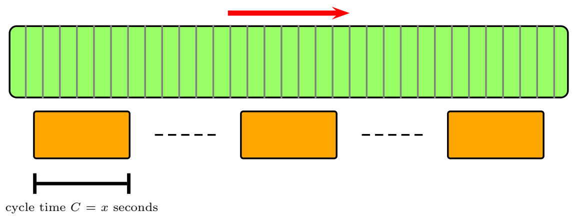
Example application: SALB

© BMW

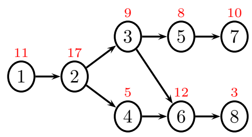
© J. Bautista & NISSAN

Problem: Simple Assembly Line Balancing (SALB)

SALB example: problem description



Tasks: Every task i has a time requirement t_i



SALB example: pheromone model

Additionally given: The maximum number *UB* of possible work stations

Objective: minimize the number of work stations needed!

1st step of applying ACO: Solution components and pheromone model

1. Solution components: we consider each possible assignment of ...
   - a task i
   - to a workstation j

   ... to be a solution component $c_{i,j}$
2. Pheromone model: we assign to each solution component $c_{i,j}$ a pheromone trail parameter $\mathcal{T}_{i,j}$ with value $\tau_{i,j}$

SALB example: solution construction

General way of working: work stations are filled one after the other (left to right)

At each iteration:

- j^*: the current work station to be filled
- T: the set of tasks ...
  1. that are not yet assigned to a work station
  2. whose predecessors are all assigned to work stations
  3. whose time requirement is such that it fits into j^*

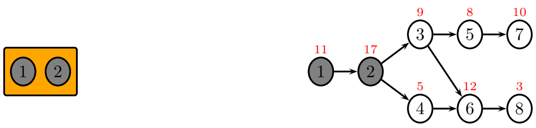
If T is empty: open a new work station

If all tasks are assigned: Stop

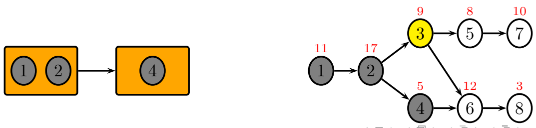
SALB example: solution construction

Assumption: cycle time is $C = 30$ seconds

Example situation 1:



Example situation 2:

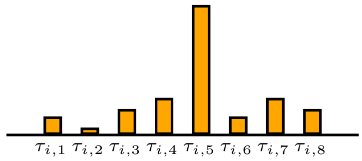


SALB example: constr. probabilities

At each construction step: how to choose a task from T?

$$\mathbf{p}(c_{i,j^*}) = \frac{\tau_{i,j^*}}{\sum_{k \in T} \tau_{k,j^*}} \qquad \forall\, i \in T$$

Disadvantage in this case:

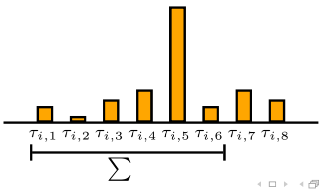


SALB example: alternative

Alternative rule: the so-called summation rule [Merkle et al., 2000]

$$\mathbf{p}(c_{i,j^*}) = \frac{\left(\sum_{h=1}^{j^*} \tau_{i,h}\right)}{\sum_{k \in T} \left(\sum_{h=1}^{j^*} \tau_{k,h}\right)} \qquad \forall\, i \in T$$

Graphical illustration: current work station is number 6



SALB example: pheromone update

Pheromone update: for example, using the iteration-best (IB) rule

pheromone evaporation	reinforcement	
$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j}$	$\tau_{i,j} \leftarrow \tau_{i,j} + \rho \cdot F(S_{ib})$	$\forall\, c_{i,j} \in S_{ib}$

where

- $\rho \in (0,1]$ is the evaporation rate
- S_{ib} is the best solution constructed in the current iteration
- $F()$ is the quality function. Again we use $F(\cdot) = \frac{1}{f(\cdot)}$

Questions?

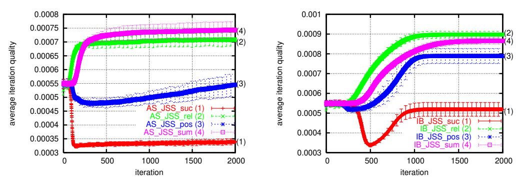
ACO: attention with the pheromones!

Note

Different pheromone models can be used to solve the same problem! The algorithm performance may vary a lot depending on the pheromone model.

Example: on the whiteboard!

Three different pheromone models for group shop scheduling (GSS)



AS-update *IB-update*

ACO: generic solution construction

Pseudo code

- $s^p = \langle\rangle$
- Determine $N(s^p)$
- **while** $N(s^p) \neq \emptyset$
  - $c \leftarrow \text{ChooseFrom}(N(s^p))$
  - $s^p \leftarrow$ extend s^p by appending solution component c
  - Determine $N(s^p)$
- **end while**

Problem

How to implement function ChooseFrom($N(s^p)$)?

ACO: generic solution construction

In a Greedy algorithm

$$c = \text{argmax}_{c' \in N(s^p)} \eta(c') \ ,$$

where $\eta : C \mapsto \mathbb{R}^+$ is a greedy function

In ACO

$$\mathbf{p}(c \mid s^p) = \frac{[\tau_c]^\alpha \cdot [\eta(c)]^\beta}{\sum\limits_{c' \in N(s^p)} [\tau_{c'}]^\alpha \cdot [\eta(c')]^\beta} \ , \quad \forall \ c \in N(s^p) \ ,$$

where $\alpha, \beta > 0$

Observations

1. α and β balance between pheromone information and greedy function
2. ACO can be applied if a constructive heuristic exists!
3. ACO can be seen as an iterative, adaptive Greedy algorithm

ACO: pheromone update

Generic pheromone update

$$\tau_c \leftarrow (1-\rho) \cdot \tau_c + \rho \cdot \sum_{\{S \in \mathcal{S}_{upd} | c \in S\}} w_S \cdot F(S),$$

Notation

- $\rho \in (0, 1]$ is the evaporation rate
- $\mathcal{S}_{upd}$ is the set of solutions used for the update
- $F : S \mapsto \mathbb{R}^+$ is the quality function. As before, when minimizing we use $F(\cdot) = \frac{1}{f(\cdot)}$
- w_S is the weight of solution S

Question

Which solutions should be used for updating?

ACO: pheromone update

Variants of the pheromone update rule

AS-update	$\mathcal{S}_{upd} \leftarrow \mathcal{S}_{iter}$ **weights:** $w_S = 1 \ \forall \ S \in \mathcal{S}_{upd}$
elitist AS-update	$\mathcal{S}_{upd} \leftarrow \mathcal{S}_{iter} \cup \{S_{\text{bsf}}\}$ (S_{bsf} is the best solution found so far) **weights:** $w_S = 1 \ \forall \ S \in \mathcal{S}_{iter}$, $w_{S_{\text{bsf}}} = e \geq 1$
rank-based AS-update	$\mathcal{S}_{upd} \leftarrow$ the $m-1$ best solutions from $\mathcal{S}_{iter} \cup \{S_{\text{bsf}}\}$ **weights:** $w_S = m - r$ for solutions from $\mathcal{S}_{iter}$, $w_{S_{\text{bsf}}} = m$
IB-update:	$\mathcal{S}_{upd} \leftarrow \text{argmax}\{F(S) \mid S \in \mathcal{S}_{iter}\}$ **weight** 1
BS-update:	$\mathcal{S}_{upd} \leftarrow \{S_{\text{bsf}}\}$ **weight** 1

ACO: algorithm variants

$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System($\mathcal{MMAS}$): characteristics

- Use of a pheromone lower bound $\tau_{min} > 0$
- Algorithm restarts (through a re-initialization of the pheromone values)
- Use of update rules IB-update and BS-update for the pheromone modification [a]

[a] Stützle, T., & Hoos, H. H. (2000). MAX–MIN ant system. Future generation computer systems, 16(8), 889-914.

ACO: algorithm variants

Ant Colony System (ACS): characteristics

- Deterministic solution construction steps with probability q

$$c = \text{argmax}_{c' \in N(s^p)} [\tau_{c'}]^\alpha \cdot [\eta(c')]^\beta$$

- Evaporation of pheromone also during the construction of a solution S:

$$\tau_c \leftarrow \gamma \tau_c + (1-\gamma)\tau_{\text{init}} \ , \forall \ c \in S \ ,$$

  where $\tau_{\text{init}} > 0$ is the initial pheromone value and $\gamma \in (0, 1]$
- Exclusive use of the BS-update for the modification of the pheromones[a]

[a] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on evolutionary computation, 1(1), 53-66.

ACO: pay attention with learning!

Artificial Intelligence Research Institute (IIIA-CSIC)

ACO search process: how is it biased?

- **Positive (wanted) bias:** Choice of (in comparison) good solutions for updating
- **Negative bias:** may originate from ...
  1. The modelling of the problem
  2. The solution construction mechanism
  3. The pheromone update

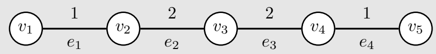
Problem

How to detect such a negative bias? A bad algorithm performance might indicate it.

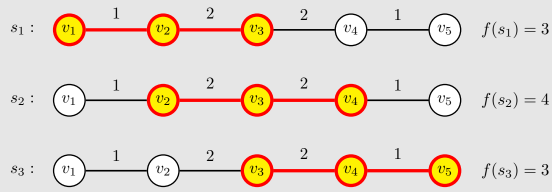
Example: negative bias

Artificial Intelligence Research Institute (IIIA-CSIC)

Example problem: 2-KCT



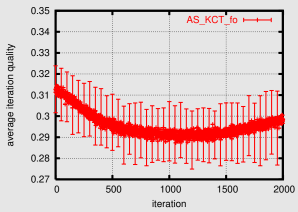
All feasible solutions



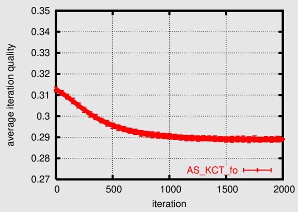
Example: negative bias

Artificial Intelligence Research Institute (IIIA-CSIC)

Average iteration quality of *Ant System*, $\rho = 0.01$



$n_a = 10$



$n_a = 1000$

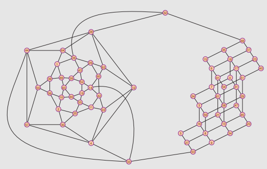
Observation

Algorithm performance decreases over time!!!

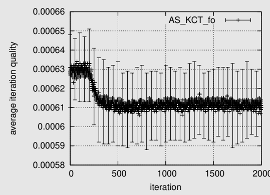
Example: negative bias

Artificial Intelligence Research Institute (IIIA-CSIC)

Application to a real instance with clusters of nodes



instance gd96c (65 nodes, 125 edges)



10 ants, $\rho = 0.1$, $k = 30$

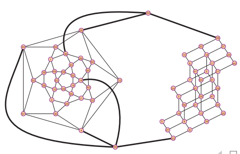
Example: negative bias

Artificial Intelligence Research Institute (IIIA-CSIC)

Definition: *Competition-balanced system (CBS)*

Let $\mathcal{P}$ be a combinatorial optimization problem. The combination of an ACO algorithm and a problem instance $P \in \mathcal{P}$ is called a CBS, if—for each partial solution s^p to P—every solution component $c \in N(s^p)$ forms part of the same number of feasible solutions.[a]

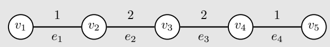
[a] Blum, C., & Dorigo, M. (2005). Search bias in ant colony optimization: On the role of competition-balanced systems. IEEE Transactions on Evolutionary Computation, 9(2), 159-174.



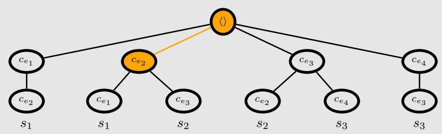
Example: negative bias

Artificial Intelligence Research Institute (IIIA-CSIC)

Example problem: 2-KCT



Search tree



Observe

Our ACO algorithm applied to this instance is **not a CBS**

ACO for continuous optimization

Reminder: continuous optimization

Given:

1. A function $f : \mathbb{R}^n \mapsto \mathbb{R}$
2. Restrictions as, for example, box constraints: $x_i \in [l_i, u_i]$

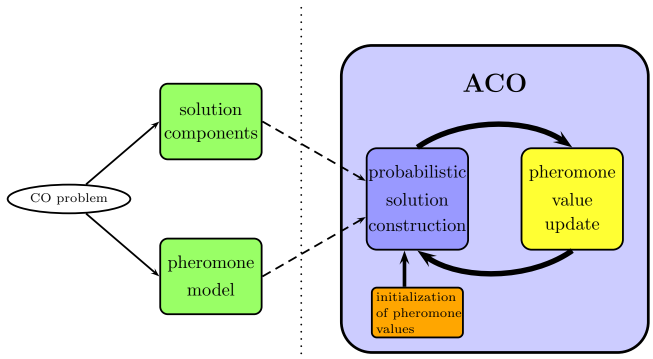
Goal: Find

$\vec{X}^* = (x_1^*, \ldots, x_n^*) \in \mathbb{R}^n$

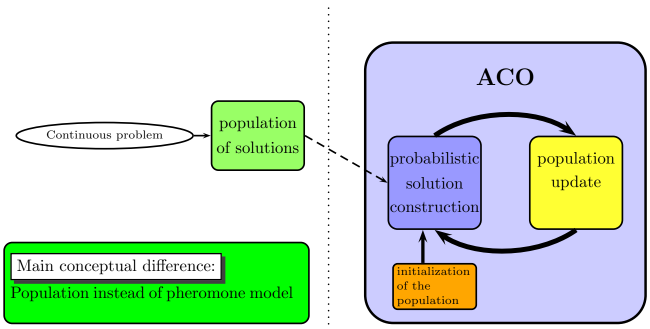
such that

- $\vec{X}^*$ fulfills all constraints
- $f(\vec{X}^*) \leq f(\vec{Y}), \forall\, \vec{Y} \in \mathbb{R}^n$

ACO for discrete problems



Continuous ACO



ACO for continuous problems

Construction/generation of a solution

Choose a value $x_i \in \mathbb{R}$ for each variable X_i, $i = 1, \ldots, n$

How to choose a value for X_i?

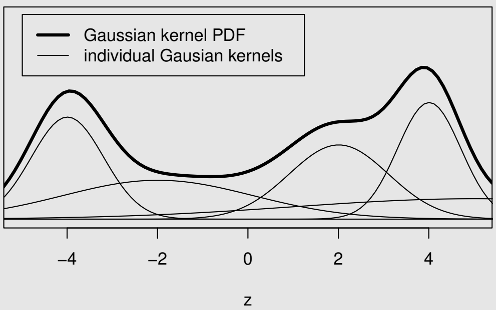
By sampling a Gaussian kernel probability density function (PDF):

$$G_i(x) = \sum_{j=1}^{k} \omega_j \left(\frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}} \right)$$

where k is the cardinality of archive P.

ACO for continuous problems

Example of a Gaussian kernel PDF



ACO for continuous problems

Problem

Sampling a PDF is not trivial

Solution to the problem

Before constructing a solution:

1. Randomly choose one of the Gaussian kernels denoted by j^*
2. Sample the Gaussian kernel j^* for all n decision variables

Methods for sampling

For example, the Box-Muller method. C++, for example, already has a method implemented.

How to choose a Gaussian kernel?

From k possible Gaussian kernels, one (j^*) is chosen according to the following probabilities:

$$\mathbf{p}_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l} \quad , \forall\, j = 1, \ldots, k$$

where:

- weights ω_j are defined as follows:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} \cdot e^{-\frac{(r_j-1)^2}{2q^2k^2}}$$

- r_j is the rank of solution j in archive P
- q is a parameter: the smaller q, the more solutions with high ranks are favored

How is the j^*-th Gaussian kernel defined?

$$g_{j^*}(x) = \frac{1}{\sigma_{j^*}\sqrt{2\pi}} e^{-\frac{(x-\mu_{j^*})^2}{2\sigma_{j^*}^2}}$$

The following values need to be determined:

1. The mean μ_{j^*}
2. The standard deviation σ_{j^*}

Mean and standard deviation

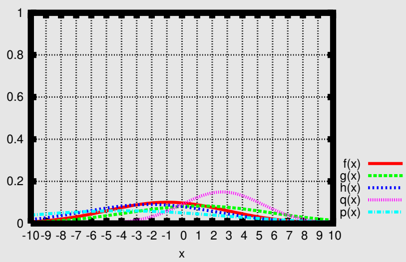
- $\mu_{j^*} = x_i^{j^*}$, where $x_i^{j^*}$ is the value of variable X_i in the j^*-th solution.
-
$$\sigma_{j^*} = \rho \left(\sum_{l=1}^{k} \sqrt{\left(x_i^l - x_i^{j^*}\right)^2} \right) / k$$

where a high value of ρ results in a slow convergence

How to deal with constraints?

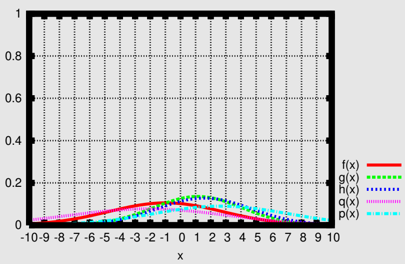
1. Repair function: Each unfeasible solution is transformed into a feasible one
2. Penalty function: Unfeasible solutions are penalized by high objective function values

Example: $f(x) = x^2$, archive size 5, 3 ants, $\rho = 2.0$



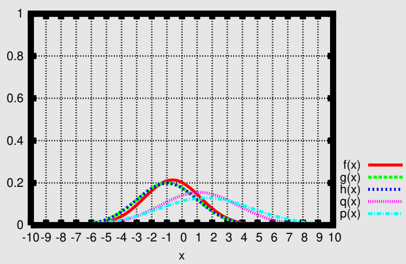
Iteration 1

Example: $f(x) = x^2$, archive size 5, 3 ants, $\rho = 2.0$



Iteration 2

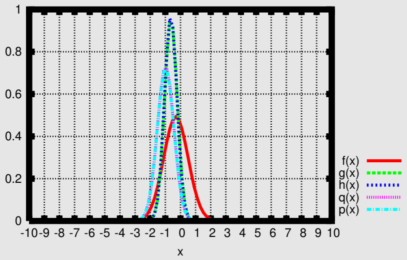
Example: $f(x) = x^2$, archive size 5, 3 ants, $\rho = 2.0$



Iteration 3

ACO for continuous problems

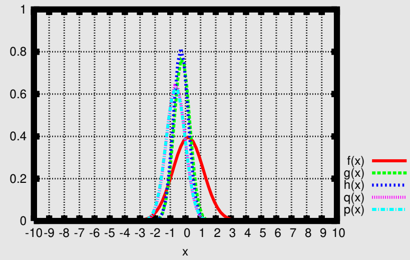
Example: $f(x) = x^2$, archive size 5, 3 ants, $\rho = 2.0$



Iteration 4

ACO for continuous problems

Example: $f(x) = x^2$, archive size 5, 3 ants, $\rho = 2.0$



Iteration 5

Questions?

Particle Swarm Optimization (PSO)

Inspiration: Social behaviour observed in animal societies

Examples:

- Flocks of birds
- Fish schools
- Gnu herds

PSO: facts

- Initially PSO was aimed at continuous optimization
- Invented by J. Kennedy and R. Eberhart in 1995.[1]
- Initial intention: Modelling the movements of flocks of birds and fish schools
- PSO deals with a swarm of particles at each iteration.
- Particles move in the solution space in the search for good solutions
- Each particle is a solution to the tackled problem
- The term particles was used because the notion of velocity was adopted.

[1] Kennedy, J., & Eberhart, R. Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks (Vol. 4, pp. 1942-1948). IEEE, 1995.

PSO: basic notations

Notation: each particle $i = 1, \ldots, m$ has a ...

- current position $\mathbf{x}_i$
- velocity $\mathbf{v}_i$
- personal best position $\mathbf{p}_i$ (memorized)

Furthermore:

- Each particle i has (or forms part of) a neighborhood $\mathcal{N}(i) \subseteq \{1, \ldots, m\}$
- $\mathbf{p}_g$ is called the neighborhood best position of i

PSO: basic algorithm

input: a continuous optimization problem in n dimensions
Generate for each particle i a random initial position $\mathbf{p}_i$, $i = 1, \dots, m$
$\mathbf{x}_i := \mathbf{p}_i$
while termination conditions not met **do**
  $\mathbf{v}_i := \mathbf{v}_i + \rho_1 \cdot (\mathbf{p}_i - \mathbf{x}_i) + \rho_2 \cdot (\mathbf{p}_g - \mathbf{x}_i)$
  $\mathbf{x}_i := \mathbf{x}_i + \mathbf{v}_i$
  if $f(\mathbf{x}_i) > f(\mathbf{p}_i)$ **then** $\mathbf{p}_i := \mathbf{x}_i$
end while
output: the best solution found

Hereby: $\rho_k = c_k \cdot r_k$, where

- c_k is the so-called acceleration coefficient
- r_k is a random number from $[0, 1]$

PSO: velocity update

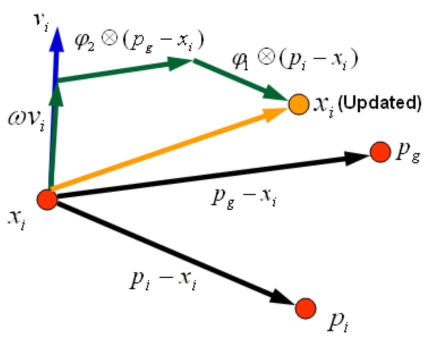
Basic update rule:

$$\mathbf{v}_i := \mathbf{v}_i + \rho_1 \cdot (\mathbf{p}_i - \mathbf{x}_i) + \rho_2 \cdot (\mathbf{p}_g - \mathbf{x}_i)$$

This rule consists of:

1. Momentum term: $\mathbf{v}_i$
   $\rightarrow$ reinforces the previous direction
2. Cognitive part: $\rho_1 \cdot (\mathbf{p}_i - \mathbf{x}_i)$
   $\rightarrow$ represents the influence of the best solution seen so far by particle i
3. Social part: $\rho_2 \cdot (\mathbf{p}_g - \mathbf{x}_i)$
   $\rightarrow$ represents the influence of the best solution seen by the neighborhood of particle i

PSO: pictorial view of update

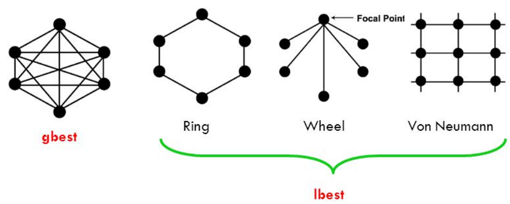


© X. Li

PSO: neighborhoods

Basic division:

1. gbest PSO: the neighborhood of each particle is the whole swarm
2. lbest PSO: neighborhoods are more restricted



PSO: neighborhoods

General observations:

- The gbest PSO converges fast, but might miss good solutions
- A lbest PSO has a slower convergence, but usually performs better
- The choice of the right neighborhood is strongly problem dependent
- Dynamic neighborhoods perform usually well, but are computationally more expensive

Systematic study: considering the degree of connectivey (k)

- **Result:** lower k favours exploration, while higher k favours exploitation

PSO: important aspects

Possible problem:

- Observation: velocities have the tendency to explode to large values
- Consequence: Particles may leave the boundaries of the search space

Possible solution:

- Velocity clamping: making use of a maximum velocity v_{max}
- Inertia weight w: $\mathbf{v}_i := w \cdot \mathbf{v}_i + \rho_1 \cdot (\mathbf{p}_i - \mathbf{x}_i) + \rho_2 \cdot (\mathbf{p}_g - \mathbf{x}_i)$

Resulting behaviour (w):

1. For $w > 1$: divergence behaviour, for $w < 1$: convergence
2. Recommendation from the literature: start with $w = 0.9$ and successively reduce the value of w to $w = 0.4$

PSO: variants

Main problem: finding a balance between exploration and exploitation

Some algorithm variants:

- Attractive and repulsive PSO (ARPSO)
  Uses different phases of attraction and repulsion between the particles
- Fitness-distance-ratio PSO (FDR-PSO)
  Encourages interaction between particles that are fit and close to each other
- Hierarchical PSO (H-PSO)
  - Organizes the particles in a dynamically changing tree structure
  - Particles are only influenced by their current father

PSO: video

PSO for binary problems

Note: first discrete PSO introduced in 1997 for binary problems

Changes: with respect to standard PSO

- The position vectors ($\mathbf{x}_i$) are binary
- The position update ($\mathbf{x}_i := \mathbf{x}_i + \mathbf{v}_i$) is re-interpreted:

$$\text{if } (r < S(v_{id})):\ x_{id} = 1. \text{ Otherwise: } x_{id} = 0$$

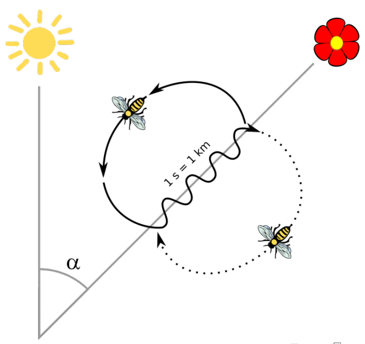
where $S()$ is a sigmoidal function, mapping all v_{id} to $[0, 1]$

Note: The velocity update can now be seen as changing the probability that bit x_i will be 1, $i = 1, \ldots, n$

Questions?

Artificial Bee Colony (ABC)

Inspiration: the way in which honey bees allocate resources for the exploitation of food sources



ABC: facts

- Initially the algorithm was introduced for continuous optimization
- Invented by D. Karaboga in 2005.[2]
- First important publication in 2007.[3]
- ABC is based on a population of solutions at each iteration.
- At each iteration, agents (artificial bees) search for better solutions in the vicinity of the solutions of the current population.

[2] Karaboga, D. An idea based on honey bee swarm for numerical optimization. Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[3] Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of global optimization, 39(3), 459-471.

ABC: different types of bees

- **Employed bees:**
  - Exploit already known food sources
  - Share information with onlooker bees (see below) by means of the waggle dance on the dance floor
- **Onlooker bees:**
  - Are recruited by employed bees for the exploitation of food sources
- **Scout bees:**
  - An employed bee that abandons an exhausted food source might become a scout bee that searches randomly for new food sources

ABC: basic algorithm

Pseudo code

input: optimization problem (on real-valued variables) in n dimensions, population size (m), parameter $l_{\text{no\_impr}}$
$P :=$ GenerateInitialPopulation(n)
$S_{\text{bsf}} :=$ best solution from P
$\mathbf{c} = (0, \ldots, 0)$ {counter for number of failed improvement attempts}
while termination conditions not met **do**
  $P :=$ EmployedBeesPhase($P, \mathbf{c}, m$)
  $P :=$ OnlookerBeesPhase($P, \mathbf{c}, m$)
  $P :=$ ScoutBeesPhase($P, m, \mathbf{c}, l_{\text{no\_impr}}$)
  $S' := \text{argmin}_{S \in P}\{f(S)\}$
  if $f(S') < f(S)$ **then** $S_{\text{bsf}} := S'$ **end if**
end while
output: S_{bsf}, that is, the best solution found

ABC: employed bees phase

Biological inspiration

This phase mimics the exploitation of food sources by employed bees.

EmployedBeesPhase($P, \mathbf{c}, m$)

input: population $P = \{S_1, \ldots, S_m\}$
for $i = 1, \ldots, m$ **do**
  $S_i' :=$ GenerateNeighbor(S_i)
  if $f(S_i') < f(S_i)$ **then** $S_i := S_i'$, $c_i := 0$ **else** $c_i = c_i + 1$ **end if**
end for

Observe

ABC algorithms normally use a greedy strategy for replacing solutions: only in case the new solution S_i' is better than solution S_i, P is modified by replacing S_i by S_i'.

ABC: onlooker bees phase

Biological inspiration

This phase mimics the recruitment of onlooker bees by means of the *waggle dance* performed by employed bees.

OnlookerBeesPhase($P, \mathbf{c}, m$)

input: population $P = \{S_1, \ldots, S_m\}$
for $i = 1, \ldots, m$ **do**
  $S_i' :=$ GenerateNeighbor(SelectSolution(P, n))
  if $f(S_i') < f(S_i)$ **then** $S_i := S_i'$, $c_i := 0$ **else** $c_i = c_i + 1$ **end if**
end for

SelectSolution(P, n)

A solution from the current population P is normally selected by roulette-wheel-selection:

$$\text{fit}(S_i) = \frac{1}{1 + f(S_i)} \quad \text{in the case of minimization}$$

ABC: scout bees phase

ScoutBeesPhase($P, m, \mathbf{c}, l_{\text{no\_impr}}$)

input: population $P = \{S_1, \ldots, S_m\}$, $\mathbf{c}$, $l_{\text{no\_impr}}$
for $i = 1, \ldots, m$ **do**
  if $c_i \geq l_{\text{no\_impr}}$ **then**
    $S_i' :=$ GenerateSolution()
    Replace S_i in P by S_i'
    $c_i := 0$
  end if
end for

GenerateSolution()

Normally, a solutions is randomly generated. However, allowed is what works!

ABC: for continuous optimization

- **Solution representation:** real-valued vectors $\mathbf{x}_i$
- **Generation of new solutions:** the **generation of the initial population** and the **generation of new solutions** is done randomly.
- **Generation of a neighbor of a solution $\mathbf{x}_i$:**
  1. Choose a solution $x_k \in P$ randomly ($x_k \neq x_i$)
  2. Sample a random value $r \in [-1, 1]$
  3. Randomly choose a dimension $j \in \{1, \ldots, n\}$
  4. Generate the neighbor: $x_i' := x_i$

$$x_{ij}' := x_{ij} + r(x_{ij} - x_{kj})$$

ABC: for combinatorial optimization

Main difference to the continuous optimization case

The generation of new solutions and the generation of neighboring solutions

Example applications

- Pulikanti, S., & Singh, A. (2009). An artificial bee colony algorithm for the quadratic knapsack problem. In International Conference on Neural Information Processing (pp. 196-205). Springer, Berlin, Heidelberg.
- Awadallah, M. A., Bolaji, A. L. A., & Al-Betar, M. A. (2015). A hybrid artificial bee colony for a nurse rostering problem. Applied Soft Computing, 35, 726-739.
- Choong, S. S., Wong, L. P., & Lim, C. P. (2019). An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. Swarm and evolutionary computation, 44, 622-635.

Questions?

Self-desynchronization in frogs

Biologists discovered:

- Male Japanese tree frogs decouple their calls
- Why do they do so?
  - The purpose of the calls is to attract females.
  - Female frogs cannot distinguish between too close calls
  - **Result:** females cannot determine the correct direction

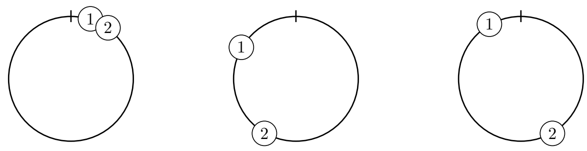
Mathematical model:

I. Aihara, H. Kitahata, K. Yoshikawa and K. Aihara. **Mathematical modeling of frogs' calling behavior and its possible applications to artificial life and robotics.** *Artificial Life and Robotics*, 12(1):29–32, 2008.

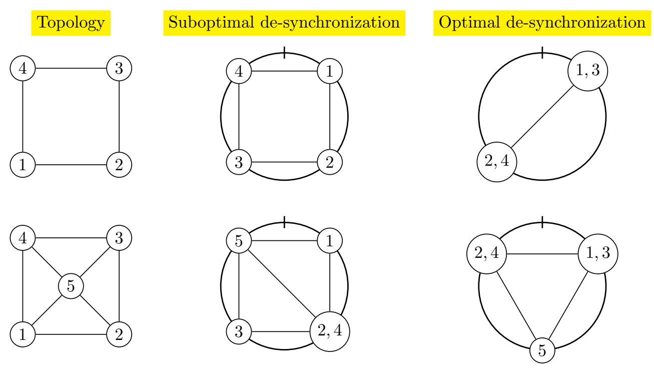
Self-desynchronization: Model

Model components:

- A set of pulse-coupled oscillators
- Some oscillators are coupled, others are independent of each other
- Each oscillator i has a phase $\theta_i \in [0, 1)$ which changes over time



Self-desynchronization: Model



Graph coloring problem

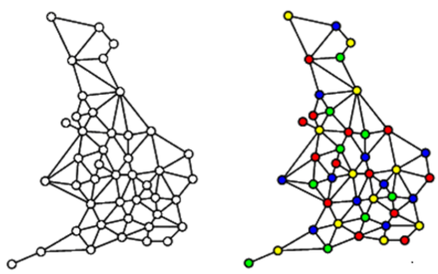


Figura 1

Goal: Use the minimum number of colours possible

A distributed algorithm

Note:

- Algorithm works with communication rounds
- **Length of such a round:** 1 time unit
- Each node can send messages to its neighbors

Each node *i* maintains ...

- $\theta_i \in [0, 1)$: a **graph coloring event** is executed by the node at time θ_i at each communication round
- $c_i \in \{0, 1, \ldots\}$: the nodes' **current color**

Graph coloring event

Artificial Intelligence Research Institute (IIIA-CSIC)

Phase I
Recalculate θ_i
Select a (new) color c_i
Send a graph coloring event message m to the neighbors

Phase II
Execute a type of distributed local search

Graph coloring event messages:

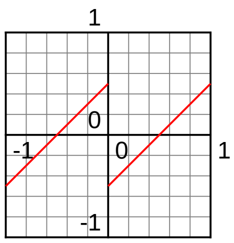
- Graph coloring event messages are stored in a separate queue M_i
- Each message m contains two values:
  - The θ-value of the sender (stored in field theta_m)
  - The senders' current color (stored in field color_m)

Recalculation of θ

Artificial Intelligence Research Institute (IIIA-CSIC)

$$\theta_i := \theta_i + \sum_{m \in M_i} \mathbf{inc}(\text{theta}_m - \theta_i)$$

Function inc():



Choosing a new color

Artificial Intelligence Research Institute (IIIA-CSIC)

$$c_i := \min\{c \in \mathbb{N} \mid \nexists m \in M_i \text{ with } \text{color}_m = c\}.$$

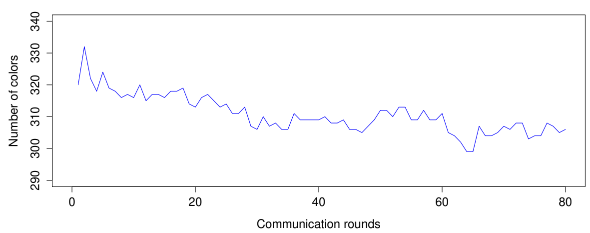
Use of the θ-values:

- They determine the order in which nodes choose a color
- This is in contrast to an existing attempt to use frogs behavior for graph coloring

Results: solution quality

Artificial Intelligence Research Institute (IIIA-CSIC)

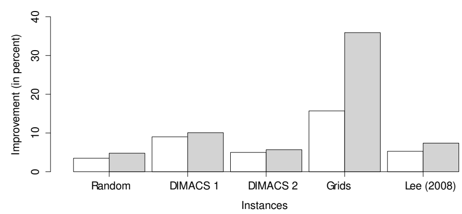
Example: DIMACS graph DSJC1000.9.col



Comparison with Finocchi Et Al.

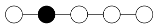
Artificial Intelligence Research Institute (IIIA-CSIC)

Average improvement (in %): over the algorithm by **Finocchi et al.**

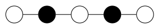


I. Finocchi, A. Panconesi and R. Silvestri. **An experimental analysis of simple distributed vertex coloring algorithms**, *Algorithmica*, 41(1), 1–23, 2005

Alg. extension: independent sets



Independent set



Maximal independent set

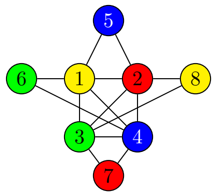
Maximum independent set

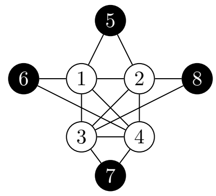
Algorithms from the literature: (for the distributed case)

- Simple distributed greedy algorithms
- Iterative self-stabilizing algorithms

Goal: finding independent sets that are as large as possible **in a distributed way**

Relationship: coloring, independent sets





Note:

- Given a feasibly colored graph, each set of nodes with the same color form an independent set
- **But:** an optimal graph coloring solution *does not necessarily contain* an optimal solution for the maximum independent set problem

Experiments

Competitors:

- Centralized greedy algorithm (just for interest)
- FruitFly: Newest iterative self-stabilizing algorithm published in the literature

Inspiration of FruitFly: Development of the fly's nervous system, when sensory organ precursor (SOP) cells are chosen

Article:

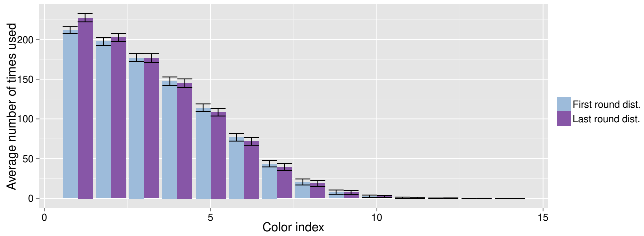
Afek, Y., Alon, N., Barad, O., Hornstein, E., Barkai, N., Bar-Joseph, Z. **A biological solution to a fundamental distributed computing problem.** *Science*, 331:183–185, 2011.

Results: random graphs ($n = 1000$)

radius (r)	Greedy	FruitFly		FrogSim		
		avg.	rounds	avg.	rounds	convergence
0.049	244.88	225.39	66.66	229.76	416.14	734.08
0.0578	190.96	176.50	115.30	180.26	414.50	758.72
0.0666	152.35	142.98	236.02	144.82	325.74	775.16
0.0754	124.53	118.74	480.64	118.82	272.97	770.88
0.0842	103.82	100.91	*1114.90*	99.55	248.03	754.76
0.093	87.82	87.19	*2562.90*	84.93	279.20	755.75
0.1018	75.42	76.72	*7014.74*	73.16	212.85	750.31
0.1106	65.61	67.91	*22063.76*	64.14	205.49	740.14
0.1194	57.84	60.60	*53523.54*	56.55	215.69	684.79
0.1282	51.53	54.60	*165323.04*	50.16	165.77	700.42
0.134	47.83	50.84	*321192.92*	46.95	160.96	678.45

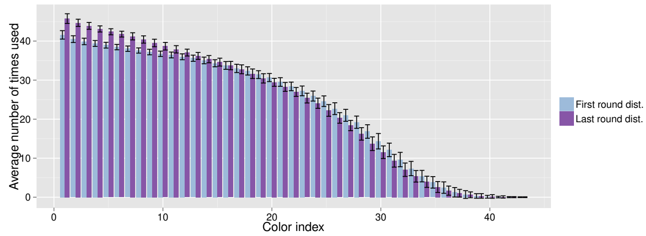
Optimization capacity (1)

Example: sparce graph (1000 nodes)



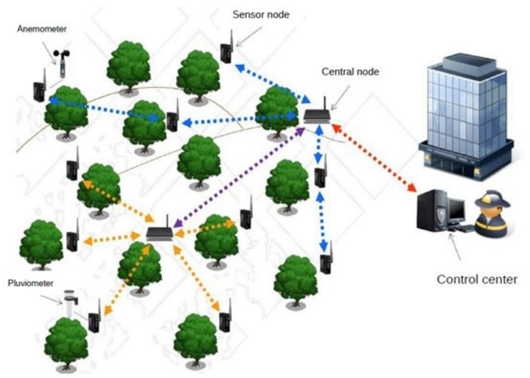
Optimization capacity (2)

Example: dense graph (1000 nodes)



Questions?

Topic: Self-synchronized duty-cycling



Self-synchronization in ants

Biologists discovered:

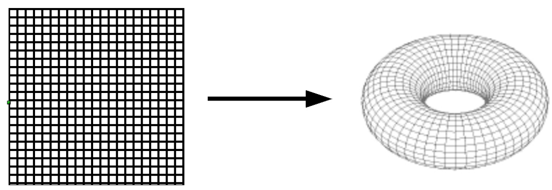
- Ant colonies show synchronized activity pattern
- Synchronization is achieved in a self-organized way: self-synchronization
- Synchronized activity ...
  1. ... provides a mechanism for information propagation
  2. ... facilitates the sampling of information from other individuals

Mathematical model:

J. Delgado and R.V. Solé. **Self-synchronization and task fulfilment in ant colonies**, *Journal of Theoretical Biology*, 205, 433–441 (2000)

Model (1)

- Each ant is modelled as an automaton
- Each automaton i can move on a LxL grid with periodic boundary conditions



Model (2)

- The state of an automaton i is described by a continuous state variable:

  $S_i(t) \in \mathbb{R}^{\geq 0}$ where t is the time step

- At each time step t, each automaton i is either active or inactive:

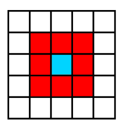
  $$a_i(t) = \Phi(S_i(t) - \theta_{act}) \text{ , where}$$

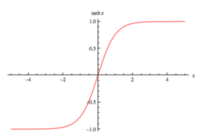
  - θ_{act}: activation threshold
  - $\Phi(x) = 1$ if $x \geq 0$. Otherwise: $\Phi(x) = 0$

Simulation

At each time step t:

1. Determine if i is active or inactive:
   - Calculate $a_i(t)$
   - If $a_i(t) = 0$: Spontaneously active i with probability $p_a > 0$ (activity level: S_a)
2. Each automaton i moves (if possible) to one of the free places in its neighborhood
3. Update the value of the state variable:





$$S_i(t+1) = \tanh(g \cdot (S_i(t) + \sum_{j \in N_i} S_j(t)))$$

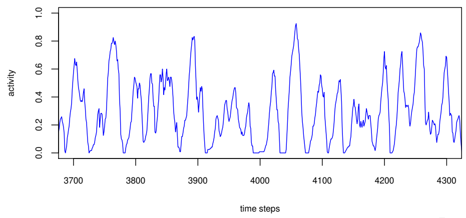
where N_i is the 8-neighborhood of the position of i

Model results/behaviour

What do we measure? mean activity of the system at time step t:

$$A(t) = \frac{1}{N}\sum_{i=1}^{N} a_i(t)$$

where N is the number of automata



Duty-cycling protocol

Note: Automata correspond to sensor nodes in a static or mobile sensor network

Organization of the protocol:

Note: during the duty-cycling (DC) phase ...

- ... all nodes are awake
- Each node executes a duty-cycling event at a **randomly chosen time**. This includes sending exactly one message.

Functionality of a sensor node

Each sensor node i ...

- has a battery with level $0 \leq b_i(t) \leq 1$
- maintains a special message queue Q_i for incoming duty-cycling messages
- is equipped with a radio antenna that allows to choose from a set $\{T^1, \ldots, T^n\}$ of n different transmission power levels
- is equipped with a solar panel that allows to collect a certain amount of energy at each time step

Content of a message $m \in Q_i$: the value of the state variable of the sender

Duty-cycling event

Calculate a_i
if $a_i = 0$ **then**
  Draw a random number $p \in [0, 1]$
  if $p \leq p_a$ **then** $S_i := S_a$ and $a_i := 1$ **end if**
end if
Determine transmission power level T_i
Compute new value for state variable S_i
Send duty-cycling message m (containing value S_i) with transmission power T_i

Transmission power levels

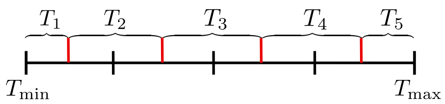
Ideal transmission power:

$$T_i := T_{\min} \cdot (1 - b_i) + T_{\max} \cdot b_i$$

where

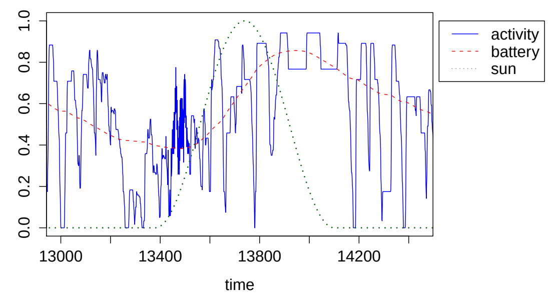
- $T_{\min}$: minimum transmission power level
- $T_{\max}$: maximum transmission power level

How did we discretize:



Results: Simulator Shawn (1)

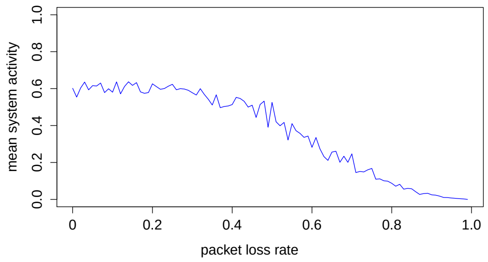
Setup: 120 sensor nodes, no packet loss is considered



Note: The average activity of the system is approx. 0.6.

Results: Simulator Shawn (2)

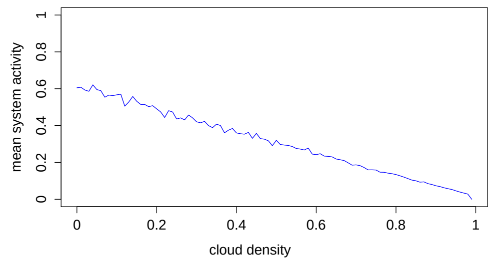
Setup: 120 sensors, different packet loss rates



Note: the system is very robust up to a packet loss rate of aprox. 0.3.

Results: Simulator Shawn (3)

Setup: 120 sensors, different cloud densities



Note: there is a linear relationship between the cloud density and the system activity

Questions?

Swarm intelligence: Quo vadis?

- Problem: swarm intelligence has attracted too many people
- As a consequence:
  1. Experienced researchers were overwhelmed with reviewing
  2. People who should have never been asked to do so did reviewing work
- Nowadays we find numerous papers in the literature that are either ...
  1. non-sense
  2. reinvent the wheel

First steps against this trend:

- Some journals (J. of Heur., Comp. & Oper. Res.) ask for algorithms to be described in metahpor-free language
- Colleagues start to expose the problem (G. Rudolph, K. Sörensen, Christian Camacho)