




UNIVERSIDAD
COMPLUTENSE
MADRID

Swarm Intelligence

Universidad Complutense 
Ismael Rodríguez, Fernando Rubio

Introduction

- ◉ **Swarm Intelligence**. Nature inspired algorithms to solve:
 - ◉ **Combinatorial** domain optimization problems (ACO, MTPSO, RFD...)
 - ◉ **Continuous** domain optimization problems (PSO, ABC, DE...)
- ◉ Many (partially) **independent entities** cooperating to find a solution.
- ◉ Individuals are relatively **homogeneous** and interact with **simple rules**
- ◉ Balance:
 - ◉ **Exploration** of the overall search space
 - ◉ **Exploitation** of the most promising areas

Introduction

- There are many (many, many, many) swarm intelligence metaheuristics.
- EvolutionaryComputationBestiary <https://github.com/fcampelo/EC-Bestiary>



- Let's try to **focus** on the **mathematical basis** of classical metaheuristics
- Most of the *original* metaheuristics only provide a *new analogy*, but are **essentially the same** as other more classical metaheuristics

Swarm Intelligence: Overall view

- Comprehensive Taxonomies of Nature- and Bio-inspired Optimization: Inspiration versus Algorithmic Behavior, Critical Analysis and Recommendations Clasificación by behavior:
 - **Differential vector movement** (207/323 => **64.09%**).
 - All population (13/323 => 4.02%). FA, CDA, ...
 - Groups based (25/323 => 7.75%). CSO, BA, ...
 - Representative based (169/323 => **52.32%**). **PSO**, ABC, ...
 - **Solution creation** (116/323 => 35.91%):
 - **Combination** (108/323 => **33.43%**). **GA**, Cuckoo Search,...
 - **Stigmergy** (8/323 => **2.48%**). **ACO**, RFD,...

Swarm Intelligence: Particle Swarm Optimization

- Inspired on social behavior of a colony of bird flocks
- Minimize fitness function $f: R^n \rightarrow R$
- Each particle has an **initial random** position and an initial random \mathbf{v} vector
- In each iteration: $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \phi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g (\mathbf{g}_d - \mathbf{x}_{i,d})$
 - **Inertia**
 - Vector towards **best local** position $\mathbf{p}_{i,d}$ found by itself
 - Vector towards **best global** position \mathbf{g}_d found by the swarm

Swarm Intelligence: Particle Swarm Optimization

- In each iteration:
 - $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \phi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g (\mathbf{g}_d - \mathbf{x}_{i,d})$
 - $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
 - **if** $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then** Update the particle's best known position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$
 - **if** $f(\mathbf{p}_i) < f(\mathbf{g})$ **then** Update the swarm's best known position: $\mathbf{g} \leftarrow \mathbf{p}_i$
 - r_p and r_g are **random** numbers between 0 and 1
 - $\omega \phi_p \phi_g$ are **parameters** of the metaheuristic. (inertia, cognitive component, and social component)

Swarm Intelligence: Particle Swarm Optimization

- Initialization:

- Randomly distributed or around candidate solution
- Velocity vectors randomly distributed or zero

- Implementation details:

- Keep positions (and velocity) inside search space
 - Closest position inside search space
 - Velocity zero or inverse (whip back)

Swarm Intelligence: Particle Swarm Optimization

- Parameter tuning:

- Classical values: social and cognitive around 2.

- Pedersen values:

<https://www.researchgate.net/profile/Mohamed-Mourad-Lafifi/post/Which-is-the-best-swarm-size-in-PSO/attachment/5b5b6f85b53d2f89289c14e1/AS%3A653084896288769%401532718981208/download/Good+Parameters+for+Particle+Swarm+Optimization.pdf>

<https://eprints.soton.ac.uk/71755/>

Swarm Intelligence: Particle Swarm Optimization

- ⊙ Variations
 - ⊙ Neighborhood (ring, adaptive topologies, etc.)
 - ⊙ Mutations, hybrid methods, simplifications, discrete version, etc.
 - ⊙ Multiobjective versions

Swarm Intelligence: Particle Swarm Optimization

- ⦿ **Your next task:**
 - ⦿ Develop **your own PSO** program
 - ⦿ Check it with classical benchmark functions (Sphere, Rastringin, Rosenbrock, etc.) [Evolutionary programming made faster](#)
 - ⦿ Compare results with different parameters. Apply statistical tests:
 - ⦿ <https://www.isa.us.es/3.0/tool/stat-service/>
 - ⦿ <http://tec.citius.usc.es/stac/ranking.html>
 - ⦿ <https://www.statskingdom.com/kruskal-wallis-calculator.html>

Swarm Intelligence: Artificial Bee Colony

- Inspired on social behavior of a colony of bees
- Minimize fitness function $f: R^n \rightarrow R$
- Three types of bees (balance local/global search):
 - **Employed bees:** work on a concrete food source (location)
 - **Onlooker bees:** select food sources depending on the dance of employed bees (quality of the solutions)
 - **Scout bees:** Randomly search for new food sources

Artificial Bee Colony: Employed Bees

- ◉ **Half** of the bees are employed bees
- ◉ **Employed** bee b_i (working on position p_i in R^n):
 - ◉ Randomly select a dimension k
 - ◉ Randomly select another bee b' (at position p_j)
 - ◉ Generate new position modifying dimension k of p_i combining it with the position of b' :
 - ◉ $p'_i = p_i$
 - ◉ $p'_{i,k} = p_{i,k} + r (p_{i,k} - p'_{j,k})$ r random value in $[-1,1]$

Artificial Bee Colony: Employed Bees

- ⦿ **Employed** bee b_i (working on position p_i in R^n):
 - ⦿ Generate **new position** modifying dimension k of p_i combining it with the position of b' :

$$p'_{i,k} = p_{i,k} + r (p_{i,k} - p'_{j,k})$$

- ⦿ **if** the fitness function **improves** in the new position
 - ⦿ Then **update** position of employed bee
 - ⦿ else **decrement trials counter** (main **parameter** of ABC)

Artificial Bee Colony: Onlooker and abandoned bees

- Onlooker bees (**half** of the total number of bees):
 - **Weighted selection** of position p of an employed bee. The most promising positions are selected with higher probability =>
exploitation
 - Work one step on that position **exactly** as an **employed** bee
- Abandoned bee: After **sc** fails in the position of an employed bee:
 - Employed bee becomes scout: **new random** position => **exploration**
 - Continue as employed bee in the new position

Artificial Bee Colony

- ⊙ **Initialization:** Randomly distributed or around candidate solution
- ⊙ **Parameter tuning:**
 - ⊙ Half of the bees are employed bees and half onlooker bees
 - ⊙ $\text{scoutingCounter} = \text{swarmSize} * \text{numberOfDimensions}$
 - ⊙ **Only one parameter** (number of bees)!!! (and number of iterations)
- ⊙ Variations
 - ⊙ Mutations, hybrid methods, discrete version, etc.
 - ⊙ Multiobjective versions

Swarm Intelligence: Differential Evolution

- ⦿ Minimize fitness function $f: R^n \rightarrow R$
- ⦿ Initialize agents at random positions of the search space
- ⦿ In each iteration, every agent tries to improve its position:
 - ⦿ **Pick three other and different agents** (a,b,c)
 - ⦿ Choose randomly one of the n dimensions (i)
 - ⦿ **Modify dimension i** of the agent, **and also (probably) other** dimensions, by combining the positions of agents a, b, and c

Swarm Intelligence: Differential Evolution

- In each iteration, every agent tries to improve its position:
 - $x_i = a_i + F (b_i - c_i)$
 - For each j of the rest of dimensions:
 - Generate random number r_j
 - If $r_j < CR$ then $x_j = a_j + F (b_j - c_j)$
 - If the fitness of the new **position improves then update** position
- **CR: crossover parameter.** CR in $[0,1]$
- **F: Differential weight.** F in $[0,2]$

Swarm Intelligence: Differential Evolution

- ⦿ Parameter tuning:
 - ⦿ Population size, number of iterations, CR, F
- ⦿ Very sensitive!!! See e.g.

<https://pdfs.semanticscholar.org/48aa/36e1496c56904f9f6dfc15323e0c45e34a4c.pdf>

Swarm Intelligence: Firefly Algorithm

- Every particle influences each other particle. Attraction depends on:
 - Quality of solution (light intensity)
 - Distance between them (light absorption of the air)
- Only difference with PSO: Every particle influences each other. Depending on parameters used, it can be converted into PSO.

Swarm Intelligence: Water Cycle Algorithm

- **Hierarchical organization.** 3 categories depending on quality of solutions:
 - 1 **sea**
 - **r rivers.** Each river *flows* towards the sea
 - **Streams** associated to rivers. Each stream *flows* towards its river. More promising rivers have more streams
- If a stream (or river) improves its river (sea), it becomes the river (the sea)
- If a stream (river) is very close to the river (sea), it is removed
- Less promising streams *evaporate*
- It *rains* to create new streams

Swarm Intelligence: Parallel implementations

- Many (partially) **independent entities** cooperating to find a solution
- Current personal computers:
 - Multicore
 - Graphics Processing Units (**GPUs**)
- E.g. PSO: $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \phi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g (\mathbf{g}_d - \mathbf{x}_{i,d})$
- Each **particle** and each **dimension** can be computed in **parallel**
 - Find appropriate granularity level

Swarm Intelligence: Parallel implementations

- Any of the previous methods can be **easily parallelized**:
 - 1) Splitting list of candidates into **nPE** (processors available) *islands*
 - 2) Each group evolves in parallel independently during **pit** iterations
 - 3) Processes communicate among them to redistribute the candidates
- This mechanism is repeated **it** times
- Total number of iterations = **it*pit**

Swarm Intelligence: Cooperation among metaheuristics

- The previous methods can cooperate in parallel:
 - 1) **Each metaheuristic** run in a different *island* (or even several islands)
 - 2) Each group evolves in parallel independently during *pit* iterations
 - 3) Processes communicate among them to *migrate* candidates
- This mechanism is repeated *it* times
- Total number of iterations = $it * pit$
- **Each method can help others to go out from local minima**

Swarm Intelligence: Conclusions and your future work

- ⦿ Swarm metaheuristics are very easy to **define** and **use**
- ⦿ Parameter tuning sensitivity... and **cheating**

Your Future work:

- **Implement** ABC and/or DE
- **Compare** results with PSO using a simple benchmark (e.g. the one presented in the paper [*Evolutionary programming made faster*](#))
- Repeat with a not so simple benchmark (e.g. the one used in [BBOB workshop](#))