Swarm Intelligence

INFO-H414

Dorigo & Birattari

Notes by Alexandre Le Mercier

Université Libre de Bruxelles

April 15, 2025

Contents

1 Introduction to Swarm Intelligence [1]

Swarm Intelligence (SI) is the study of how large groups of relatively simple agents (e.g., social insects like ants and bees, or birds and fish) can produce complex collective behavior through local interactions and self-organization. In engineering, SI techniques are used to solve optimization problems, cluster data, and coordinate multi-robot systems, among other applications. This summary reviews the key concepts of Swarm Intelligence, including self-organization and stigmergy, and then presents three main algorithmic frameworks inspired by biological swarms: ant-based clustering, Particle Swarm Optimization, and Ant Colony Optimization.

1.1 Key Concepts in Swarm Intelligence

1.1.1 Self-Organization

A defining characteristic of SI is *self-organization*, in which high-level patterns emerge from simple local interactions among agents. Typical ingredients of self-organization include:

- **Positive feedback:** Small, random fluctuations get amplified (e.g., pheromone reinforcement in ants).
- **Negative feedback:** Limits growth and avoids system saturation (e.g., limited availability of foragers).
- **Multiple interactions and randomness:** Agents interact frequently, and small stochastic decisions at the local level can produce sophisticated global patterns.

1.1.2 Stigmergy

Stigmergy is a form of indirect coordination in which agents affect the environment in ways that influence subsequent behavior of other agents. In natural systems, it often takes the form of pheromone trails left by ants or local modifications to a wasp's nest that guide future construction. Artificial stigmergy is similarly defined as indirect agent communication via shared environmental states.

1.2 From Nature to Algorithms

Swarm Intelligence is motivated by observing a social behavior in nature, modeling it, and then adapting that model to solve a practical problem. Three illustrative examples are:

- **Cemetery organization in ants** $\rightarrow$ *Data clustering algorithms.*

- **Flocking in birds** $\rightarrow$ *Continuous optimization (Particle Swarm Optimization).*
- **Foraging in ants** $\rightarrow$ *Shortest-path optimization (Ant Colony Optimization).*

1.3 Ant-Based Clustering and Sorting

A classic example involves *cemetery organization*: in nature, ants gather dead nest-mates into organized clusters. A simplified model places ants on a 2D grid with scattered "items." Each ant probabilistically picks up or drops items based on a measure of similarity in its neighborhood.

1.3.1 Basic Model

Let $f(o_i)$ denote the perceived fraction of items near item o_i. Two threshold parameters k_1 and k_2 control pick-up and drop probabilities:

$$P_{\text{pickup}}(o_i) = \left(\frac{k_1}{k_1 + f(o_i)}\right)^2, \quad P_{\text{drop}}(o_i) = \left(\frac{f(o_i)}{k_2 + f(o_i)}\right)^2.$$

Here, $f(o_i)$ can be computed by examining neighboring items within a small window around o_i, often using a distance function. For instance, if each data point o_i has attributes $(x_{i,1}, \ldots, x_{i,n})$, then:

$$d(o_i, o_k) = \frac{1}{n} \sum_{j=1}^{n} \left| x_{i,j} - x_{k,j} \right|.$$

When many similar items are nearby, the perceived fraction $f(o_i)$ grows, and the ant is more likely to drop rather than pick up. Over time, this yields emergent clusters of similar items.

1.4 Particle Swarm Optimization (PSO)

Inspired by bird flocking and fish schooling, Particle Swarm Optimization treats candidate solutions as "particles" moving through the space of possible solutions. Particles *fly* toward areas of higher quality, guided by their own best known position and by the best known position found by their neighbors.

1.4.1 Algorithmic Framework

- **Initialization:** Randomly position N particles in the search space and assign random velocities.
- **Evaluation:** Each particle i has a current position $\mathbf{x}_i^t$ and a velocity $\mathbf{v}_i^t$ at iteration t. A fitness function f evaluates the quality of $\mathbf{x}_i^t$.

- **Personal and Neighborhood Bests:** Each particle stores its best position so far, denoted $\mathbf{pbest}_i^t$. Additionally, it has access to either the global best $\mathbf{gbest}^t$ or a local best $\mathbf{lbest}_i^t$ within its neighborhood.

- **Velocity and Position Updates:**

$$\mathbf{v}_i^{t+1} = \underbrace{w\,\mathbf{v}_i^t}_{\text{inertia}} + \underbrace{\phi_1 \mathbf{U}_1^t \big(\mathbf{pbest}_i^t - \mathbf{x}_i^t\big)}_{\text{personal influence}} + \underbrace{\phi_2 \mathbf{U}_2^t \big(\mathbf{lbest}_i^t - \mathbf{x}_i^t\big)}_{\text{social influence}} .$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1},$$

  where w is the *inertia weight*, ϕ_1 and ϕ_2 are acceleration coefficients, and $\mathbf{U}_1^t, \mathbf{U}_2^t$ are random vectors in $[0,1]^n$.

- **Repeat:** Evaluate new positions, update personal and neighborhood bests, and iterate until a stopping criterion is met.

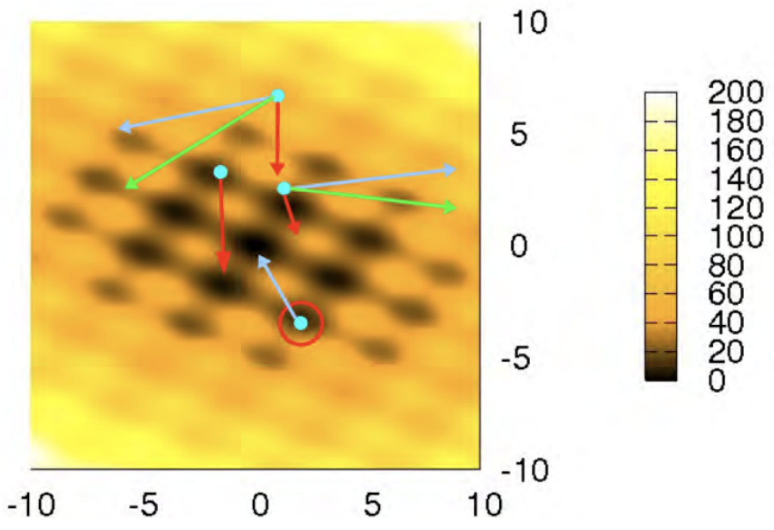

Figure 1: Illustration of PSO: Acceleration are represented by red arrows, previous velocity with blue arrows, and new velocity with green arrows.

PSO effectively balances *exploration* (searching broadly) and *exploitation* (refining known good solutions) by tuning w, ϕ_1, and ϕ_2.

1.5 Ant Colony Optimization (ACO)

Foraging ants deposit pheromones along paths leading to food sources, and subsequent ants tend to follow stronger pheromone concentrations. This *positive feedback* can lead ants to converge on the shortest path. In *Ant Colony Optimization*, artificial ants construct solutions (e.g., paths) on a graph and reinforce edges used in good solutions by depositing synthetic "pheromone," guiding future ants.

- **Initialization:** Set pheromone levels on edges to some initial value.
- **Construct Solutions:** Each ant builds a solution step by step, probabilistically choosing edges based on pheromone strength and local heuristics (e.g., inverse of distance).
- **Pheromone Update:** Once solutions are built, pheromone is *evaporated* (a negative feedback) and *deposited* on edges used by the best ants (positive feedback).
- **Iteration and Convergence:** Over many iterations, edges used in high-quality solutions accumulate higher pheromone, biasing future ants toward promising solutions.

1.6 Conclusions

Swarm Intelligence provides powerful insights and strategies for distributed problem-solving. Self-organization, stigmergy, and local interactions often lead to robust and scalable behaviors. The three main algorithmic families highlighted here demonstrate how these natural processes inspire solutions to clustering, continuous optimization, and shortest-path or combinatorial optimization. Together, they illustrate the broad applicability and effectiveness of Swarm Intelligence in computational and engineering domains.

2 Ant Colony Optimization for Routing and Combinatorial Problems [2]

2.1 Introduction

Ant Colony Optimization (ACO) emerged as a powerful nature-inspired technique for solving distributed, dynamic, and often NP-hard optimization tasks. Its key insight comes from *stigmergy* in real ant colonies, where foragers collectively discover short paths to food sources by depositing and following pheromone trails. In ACO, this mechanism is adapted to graph-based problems such as the Traveling Salesman Problem (TSP), network routing, and scheduling, providing both robust exploration and adaptive exploitation of promising solutions.

This chapter first lays out the *mathematical foundations* of ACO and explains how artificial ants build and reinforce solutions. We then illustrate potential pitfalls and how ACO overcomes them. Next, we contrast the *static shortest path problem* with the more challenging *dynamic routing problem*, clarifying why the latter often benefits from a swarm-based approach. We detail *AntNet*, a distributed algorithm for telecommunication routing, before discussing classical *ACO variants* (including *ACS*) and a specialized algorithm called *HAS-SOP* for the Sequential Ordering Problem. We conclude with an overall perspective on how ACO's design principles yield state-of-the-art performance in both routing and combinatorial optimization.

2.2 Mathematical Foundations of ACO

The generic ACO framework uses a colony of m *ants* to iteratively build solutions on a *construction graph*:

1. **Representation:** Each edge (i, j) has a real-valued *pheromone* $\tau_{ij}(t)$ indicating how desirable it is, and a *heuristic* value $\eta_{i,j}$ (e.g., $\frac{1}{\text{distance}(i,j)}$).

2. **Solution Construction:** An ant at node i chooses its next node $j \in \mathcal{N}_i$ (the feasible neighbors) with probability

$$P_{i,j}(t) \;=\; \frac{\left[\tau_{i,j}(t)\right]^{\alpha}\,\left[\eta_{i,j}\right]^{\beta}}{\sum_{k\in\mathcal{N}_i}\left[\tau_{i,k}(t)\right]^{\alpha}\,\left[\eta_{i,k}\right]^{\beta}},$$

   where $\alpha, \beta > 0$ control the relative influence of the pheromone trail vs. the heuristic.

3. **Pheromone Update:** After each ant completes a solution (e.g., a path or tour), edges used in *high-quality* solutions receive additional pheromone:

$$\tau_{i,j}(t+1) \;=\; (1-\rho)\,\tau_{i,j}(t) \;+\; \Delta\tau_{i,j},$$

where $0 < \rho < 1$ is the *evaporation rate* controlling pheromone decay, and $\Delta\tau_{i,j}$ depends on each ant's solution quality (e.g., $\Delta\tau_{i,j} = \sum \frac{1}{L_k}$ if edge (i, j) is in ant k's solution of length L_k).

This interplay of *positive feedback* (reinforcement on good edges) and *negative feedback* (evaporation) balances *exploration* of new paths with *exploitation* of known good ones.

Self-Reinforcing Loops and Their Mitigation

A known drawback in naive implementations of the pheromone mechanism is the possibility of *self-reinforcing loops* (Fig. 2). In such a loop, an initially random choice can lead many ants to favor a suboptimal edge. As pheromone accumulates on that edge, it continually draws in more ants, compounding the bias. To mitigate this pitfall, most ACO algorithms combine several strategies:

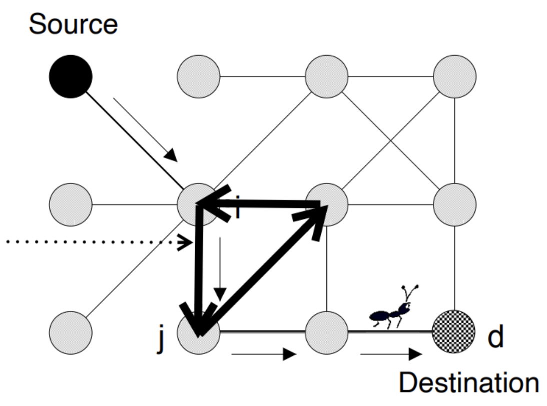

Figure 2: A conceptual diagram of a self-reinforcing loop: early random events cause a suboptimal path to become overexploited, attracting further ants and accumulating ever more pheromone.

- **Delayed Pheromone Updates:** Rather than depositing pheromone during forward travel (*forward updating*), ants often wait until an entire solution is constructed (*backward updating*) before reinforcing the used edges. This avoids amplifying random early decisions, thereby reducing the chance that a poorly chosen edge gets locked in too soon.

- **Evaporation:** By multiplying pheromone levels by $(1 - \rho)$ at each iteration, edges that do not receive new reinforcement gradually lose pheromone. This constant decay helps remove the influence of spurious paths and prevents any one edge from dominating the search indefinitely.

- **Heuristic and Local Search:** Finally, domain-specific heuristics (e.g., using distance to guide ants) and optional local search steps ensure that ants do not rely solely on pheromone values. If the pheromone is misleading, the heuristic or local search can drive ants to discover higher-quality edges, breaking the loop.

2.3 Shortest Path and Routing Problems

2.3.1 Static Shortest Path

Classical *shortest path* instances (with no constraints) are solvable in polynomial time via Dijkstra's or Bellman-Ford algorithms. However, real-world variants often add constraints or dynamic edge costs (e.g., traffic variations, time windows, capacities) that can make them NP-hard or at least computationally difficult. ACO excels when standard deterministic methods cannot easily adapt to such complexities.

2.3.2 Adaptive Routing in Networks

Routing problems in telecommunications must cope with multiple sources and destinations, unpredictable traffic, and constantly changing link costs. While the "all-pairs shortest path" is polynomial, *adaptive* routing is more challenging because it requires continuous recalculation as conditions change. ACO's decentralized decision-making and pheromone updates allow dynamic adaptation, making it an appealing choice for network routing.

2.4 Real vs. Artificial Ants

ACO borrows fundamental ideas from real ants, pheromone-based stigmergy, distributed search, and short-path bias, but diverges in crucial ways:

Shared Traits

- *Distributed, colony-based control.*
- *Positive feedback* through pheromone reinforcement.
- *Stochastic exploration* with local awareness (no centralized blueprint).

Distinct Adaptations

- *Discrete problem representation*: Artificial ants work on graph edges rather than continuous terrain.
- *Solution-quality-based pheromone deposit*: Real ants deposit pheromone at roughly constant rates, whereas artificial ants deposit amounts scaled by how "good" their solutions are.

- *Memory and heuristics*: Artificial ants can retain visited-node histories and exploit domain-specific heuristics or local search steps.

2.5 AntNet: ACO for Adaptive Routing

AntNet (Dorigo & Di Caro) is a prominent ACO algorithm aimed specifically at *dynamic, distributed network routing*. Each node in the network periodically launches *Forward Ants (F-ants)* to gather path-quality information for random destinations.

2.5.1 Forward Ants (F-ants)

- **Movement Rule:** A forward ant from node i going to destination d chooses the next hop j with probability

$$P_{i,j,d}(t) \;=\; \frac{\tau_{i,j,d}(t) \;+\; \alpha\, l_j}{1 \;+\; \alpha\,(|\mathcal{N}_i| - 1)},$$

  where $\tau_{i,j,d}(t)$ is the pheromone for "destination d" on edge (i,j), l_j is a local heuristic (e.g. a normalized queue length), and α balances pheromone vs. heuristic.

- **Data Gathering:** Each F-ant records the *path* it takes and the *time intervals* for each hop, effectively building a trip-time profile from i to d.

2.5.2 Backward Ants (B-ants)

Once an F-ant reaches destination d, it becomes a *Backward Ant*:

- **Priority Transmission:** B-ants are queued with higher priority to rapidly propagate their collected routing info.

- **Pheromone Update:** At each node along the reverse path, the B-ant updates the pheromone table for (current node, next hop, d) based on how quickly it reached d compared to historical statistics (mean μ and standard deviation σ). If T is the time from i to d:

$$\Delta\tau_{i,j,d} \;=\; 1 - f\!\left(\tfrac{T-\mu}{\sigma}\right), \qquad \tau_{i,j,d}(t+1) \;=\; (1-\rho)\,\tau_{i,j,d}(t) + \rho\,\Delta\tau_{i,j,d},$$

  where f is a sigmoid in $[0,1]$ and ρ is the evaporation factor.

2.5.3 Distributed and Asynchronous Operation

AntNet does not rely on global synchronization; each node independently launches ants and updates its routing table. This approach scales to large, complex networks and adapts to changes without needing a centralized coordinator.

2.6 Experimental Setup and Key Results

AntNet has been evaluated under varied scenarios, typically involving:

- **Network Topologies:**

  6×6 **grid (36 nodes):** A regular lattice often used to test large path diversity.

  SIMPLEnet (8 nodes): Minimal topology for rapid iteration and debugging.

  NSF net & NTT net: More complex, realistic patterns drawn from real telecom networks.

- **Traffic Loads:** Ranging from moderate traffic (low load) to near-saturation levels, plus transient overload conditions.

- **Performance Measures:**

  - *Throughput (bits/sec)*: measures **quantity of service** (packets delivered per unit time).
  - *Average delay (sec)*: measures **quality of service** (mean packet delivery time).

- **Comparisons:** OSPF, SPF, Q-routing, PQ-R, and a "Daemon" approach with near-omniscient routing decisions.

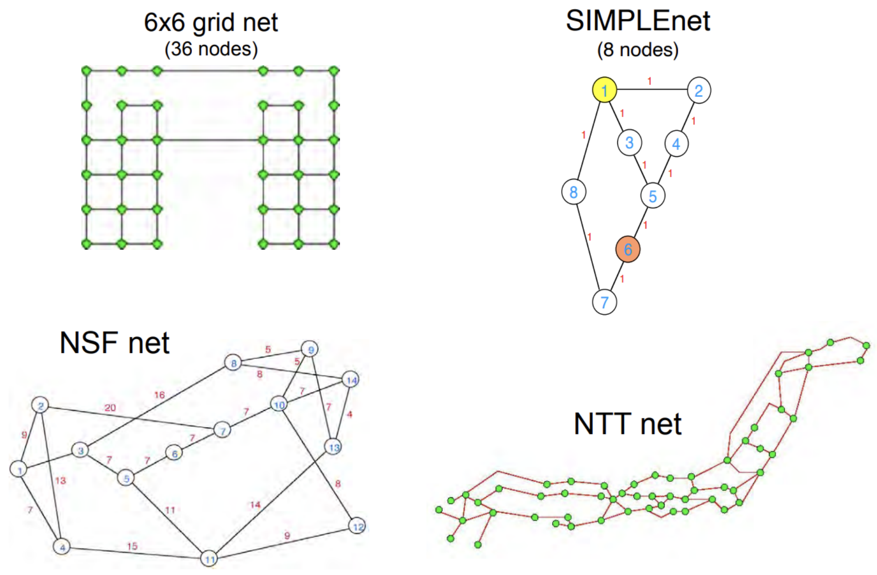

Figure 3: Common reference topologies for AntNet: a 36-node grid, SIMPLEnet, NSF net, and NTT net.

Conclusions from Experiments

1. *Low Load:* All algorithms performed similarly, as congestion is minimal.

2. *Near Saturation (High Load):* AntNet achieved higher throughput and lower average delays than the alternatives, indicating superior adaptation.
3. *Transient Overloads:* Under sudden spikes in traffic, AntNet quickly adjusted, maintaining top performance metrics for both throughput and delay.
4. *Overhead:* Control ants introduced negligible overhead compared to the gains in routing efficiency.

2.7 Extensions of ACO

While the foundational *Ant System* introduced a basic blueprint, various extensions refine or accelerate convergence:

2.7.1 Elitist AS (EAS)

Awards extra pheromone to the *best-so-far* solution each iteration, hastening path reinforcement.

2.7.2 Rank-Based AS (ASrank)

Ranks all solutions from best to worst and distributes pheromone proportionally to an ant's rank, giving top solutions a stronger influence.

2.7.3 Max–Min Ant System (MMAS)

Restricts pheromone to an interval $[\tau_{\min}, \tau_{\max}]$, typically letting only the *best-so-far or iteration-best* ant deposit pheromone. The system resets pheromone if stagnation is detected.

2.7.4 Ant Colony System (ACS)

ACS focuses on:

- *State Transition Rule:* Combines deterministic "exploitation" with stochastic "biased exploration" controlled by a threshold q_0.
- *Local Pheromone Update:* Immediately after an ant chooses an edge, a small *local evaporation* step is applied to that edge, encouraging more uniform exploration across iterations.

- *Global Update:* Only the *best tour* receives final pheromone reinforcement, concentrating learning on a single top solution each cycle.

2.8 HAS-SOP: Hierarchical ACO for the Sequential Ordering Problem

The *Sequential Ordering Problem (SOP)* asks for a Hamiltonian path (not a cycle) from a special *Start* node to an *End* node, visiting all intermediate nodes exactly once while respecting precedence constraints. Such constraints arise in supply-chain scheduling, flexible manufacturing, and routing with pickup–delivery restrictions.

- **HAS-SOP** employs hierarchical ACO layers and local search to refine feasible paths respecting the precedence constraints.
- It has demonstrated improvements or matches to known upper bounds in various TSPLIB-based SOP instances, highlighting the adaptability of ACO to diverse combinatorial tasks.

2.9 Conclusions

ACO's decentralized and self-organizing nature, paired with its blend of *pheromone reinforcement* and *evaporation*, makes it remarkably versatile for both *dynamic routing* and *NP-hard* combinatorial optimization. By carefully avoiding self-reinforcement traps, incorporating domain heuristics, and applying local search, ACO systems consistently demonstrate robust performance.

- **Routing (AntNet):** Adapts routing decisions to real-time network conditions, achieving high throughput and low delay, especially under heavy or rapidly changing traffic.
- **Combinatorial Optimization (ACS, HAS-SOP, etc.):** Effectively tackles classical problems like TSP and SOP, leveraging reinforcement to converge on high-quality solutions quickly.

Through continuing refinements, such as *Max–Min Ant System* or *Rank-Based AS*, ACO remains a leading metaheuristic for complex search spaces. Its success in both theoretical benchmarks and practical applications underscores the power of *swarm intelligence* as a design paradigm for distributed problem-solving.

3 Swarm Robotics [3]

3.1 Introduction

This chapter provides an in-depth overview of *swarm robotics*, a field studying how to design and control large groups of relatively simple robots so that effective collective behavior emerges. The goal is twofold: (i) harness local interactions (among robots and with the environment) to achieve global coordination, and (ii) maintain desirable properties such as fault tolerance, scalability, and flexibility. We begin by clarifying the core motivations behind swarm robotics, then progressively move through the main control paradigms, algorithmic strategies, and illustrative case studies.

3.2 Foundations of Swarm Robotics

3.2.1 Motivation and Definition

Swarm robotics derives from the broader area of *swarm intelligence*, which seeks to replicate the self-organizing abilities seen in natural swarms (e.g., ants, bees). The essential characteristics include:

- **Large number of robots**: Individual units (often termed *agents*) are relatively simple and inexpensive.
- **Decentralized control**: No single robot has global knowledge; instead, decisions are made locally.
- **Local interactions and communication**: Robots rely on neighbor-based sensing and message exchange.
- **Fault tolerance and scalability**: The system gracefully handles partial failures and can grow or shrink in size without a complete redesign of control strategies.

These goals are difficult to achieve with traditional *centralized* architectures, which often suffer from a single point of failure and limited scalability.

3.2.2 The Micro-Macro Link Problem

A key difficulty in swarm robotics is bridging the gap between individual (micro) rules and the desired group (macro) outcome. While the collective effect is what ultimately matters (e.g., gathering resources, exploring terrain), any single robot only follows local control policies.

Determining how to program these local policies so that emergent group behavior remains both predictable and robust is known as the *micro–macro link problem*.

3.3 Core Design Paradigms

3.3.1 Self-Organization via Local Behaviors

Because no central coordinator exists, robots in a swarm must *self-organize* through simple rules:

- **Proximity-based cues:** Robots detect obstacles, neighbors, or holes by short-range sensing.
- **Stigmergic feedback:** Similar to ants laying pheromones, robots sometimes indirectly coordinate through environment modifications or signals that persist momentarily in the world.
- **Redundancy and distributed control:** Many robots carry out tasks in parallel; if one fails, others can fill its role.

3.3.2 Programming Methodologies

Two broad categories of methods are often used to develop control strategies:

1. **Bio-inspired or machine learning approaches:** Neural networks or evolutionary algorithms are evolved in simulation before being deployed on real hardware.
2. **Hand-crafted controllers:** Designers carefully specify sensing and actuation rules, drawing on known behaviors (e.g., obstacle avoidance, formation control).

While bio-inspired methods can discover unexpected yet effective solutions, hand-crafted approaches can be more transparent to analyze.

3.4 Case Studies and Key Algorithms

In what follows, we detail core case studies illustrating how local behaviors can solve common tasks in swarm robotics.

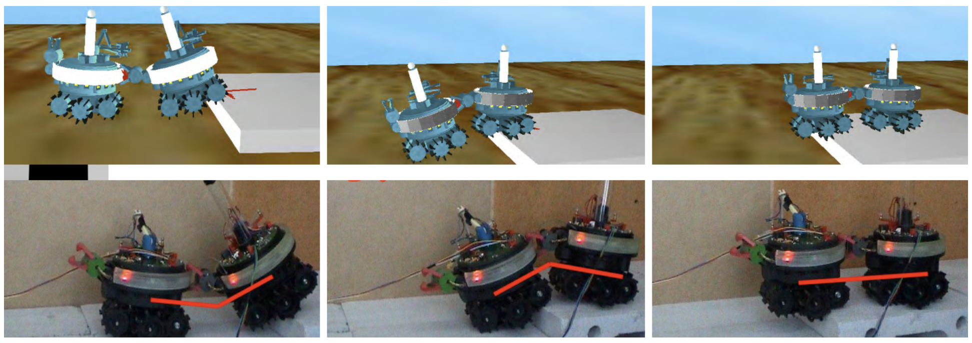
Figure 4: Two s-bots cooperating to overcome an obstacle tested in a detailed simulation (up) and realistic environment (down).

3.4.1 Coordinated Motion

Objective and Setup

A group of robots (s-bots) is connected (physically linked) in a swarm-bot formation but initially oriented randomly. They must move together in a common direction without a central command.

Traction Sensor and Control Principle

Each robot is equipped with a *traction sensor* mounted on the chassis-turret connection. When the group starts moving, mismatched directions induce pulling/pushing forces on individual turrets. By reading these forces locally, a robot can *steer* toward the average motion of neighbors (see Fig. 4).

Evolutionary Optimization

A single-layer perceptron was *evolved* to produce wheel-speed commands based on sensor inputs. The fitness function for an individual controller often emphasizes:

- *Distance covered* by the swarm-bot as a whole.
- *Uniformity of direction* (i.e., minimal tangling of forces).

One typical formulation of the group-level fitness is

$$F = \frac{\|X(T) - X(0)\|}{D},$$

where $X(t)$ is the center of mass of the entire swarm-bot at time t and D is the maximum possible distance over the experiment's duration. Evolution in simulation successfully yields controllers which then transfer well to real hardware.

3.4.2 Hole Avoidance

Motivation

For ground robots, *falling into holes* represents a major hazard. When physically connected, one robot's misstep can compromise others.

Sensing and Reflex Signaling

In addition to the traction sensor, each robot has:

- *Ground sensors* to detect edges or holes.
- *Microphones & speakers* to communicate urgent signals (e.g., a single robot detecting a hole may produce a warning tone).

This warning triggers a collective avoidance maneuver, helping all connected robots reorient away from danger.

Evolution of Communication

An evolutionary algorithm again *shapes* how robots interpret and act on acoustic signals. The direct benefits include faster responses—rather than each robot independently detecting the hole, they can heed the signal of the earliest detector. Experiments confirm robust avoidance on both simulated and real platforms.

3.4.3 Self-Assembly

Concept

When tasks demand greater pulling or pushing force (or bridging over large gaps), robots can *self-assemble* into bigger structures. Physical connectors (e.g., grippers or docking rings) allow them to lock together in ad hoc shapes.

Adaptive Morphologies

In practice, self-assembly provides:

- **Increased stability:** A single robot may topple on steep terrain, but a connected group remains upright.
- **Enhanced capabilities:** Larger formations can cross bigger obstacles.

Such transformations happen *on demand*, making the approach extremely flexible.

3.4.4 Cooperative Transport

Motivation

Cooperative transport is inspired by ants carrying large prey. Robots attach to an object and collectively move it to a desired location, possibly weaving around obstacles. Key design elements include:

- **Shared directionality:** The group must push or pull in a roughly unified heading.
- **Adaptive formation control:** Individual angles and docking positions may change as the environment changes.

Implementation

After *coordinated motion* is learned, the same principles are extended: the cargo effectively becomes part of the swarm-bot. Feedback from traction sensors helps each robot gauge the desired movement relative to the group.

3.4.5 Path Formation

Inspiration

In ants, pheromone trails guide nestmates to food. Here, robots cannot lay chemical trails, but can act as *mobile beacons*, turning on lights or signals to mark a path from a starting object to a goal region.

Emergent Trail Creation

Some robots remain idle to maintain a chain of markers, while others shuttle back and forth carrying items. As tasks continue, these ephemeral “robotic trails” can adjust in real time (e.g., if a segment fails or a shorter path emerges).

3.5 Comparative Considerations

Choice of Topologies

Numerous experimental scenarios can be considered: square arenas with open borders (for hole avoidance), modular swarm-bots with 3 to 6 connected robots (for coordinated motion or bridging experiments), and advanced "Swarmanoid" setups with three different robot types. Each topology is chosen to approximate real-world constraints (e.g., a research lab environment with open edges) while still being tractable for systematic study.

Performance Metrics

Typical metrics include:

- *Distance traveled* or *time to complete* a task.
- *Survivability*, such as successfully avoiding holes or obstacles.
- *Scalability*, measured by how well a strategy extends to larger groups.

Throughput and delay metrics also appear in more complex routing or transport tasks, reflecting how quickly items are moved compared to total resources expended.

3.6 Extended Approaches

3.6.1 Morphological Control and Fault Detection

Shape-Shifting for Multiple Tasks

A swarm might need to alternate between *pushing a ball up a ramp*, *forming a bridge*, or *traversing a confined passage*. Each requires a distinct group shape. By dynamically re-attaching or detaching, robots reconfigure on-the-fly.

Firefly-Inspired Fault Detection

In one demonstration, each robot blinks a "heartbeat" signal. If a robot is broken or detects an internal fault, it can stop blinking. Neighbors interpret the silence as a signal to isolate the faulty unit, thus ensuring the swarm remains robust.

3.6.2 Towards Swarmanoid

The *Swarmanoid* experiment showcases heterogeneous robots:

- **Foot-bots**: Ground-based with wheels.
- **Hand-bots**: Equipped with climbing or grasping capabilities.
- **Eye-bots**: Aerial or elevated vantage for overhead monitoring.

These specialized roles allow the swarm to function seamlessly in three-dimensional spaces (e.g., a room with walls and shelves). By self-organizing, the swarm adapts its distribution of capabilities (mobile, climbing, scanning) to efficiently perform a variety of tasks.

3.7 Conclusion

Swarm robotics embodies the principles of *distributed control, local sensing, and emergent organization*. Through case studies of coordinated motion, hole avoidance, self-assembly, cooperative transport, and path formation, this chapter illustrates how diverse tasks can be tackled by simple robots acting collectively. The evolutionary methods and careful sensor-actuator design emphasize how **local interactions** give rise to global coherence without a central controller.

In addition, advanced directions such as morphological reconfiguration and multi-robot heterogeneity (*Swarmanoid*) open further applications, solidifying the promise that *fault-tolerant, scalable, and flexible* robot swarms can handle complex, real-world scenarios. This fundamental framework sets the stage for deeper investigations into optimization, theoretical guarantees, and expanded deployment in robotics research.

4 Division of Labor and the Threshold Model [4]

4.1 Overview

In this chapter, Mauro Birattari explores the principles underlying the division of labor (DoL) and introduces the *threshold model* as a mechanism for self-organized task allocation. The discussion spans examples from human societies and insect colonies, and extends these concepts to robotic swarms. Central to the chapter is the notion that plasticity, or the ability to reassign tasks dynamically, is key to resilience and efficiency in complex systems.

4.2 Division of Labor in Social Systems

Human Societies:

DoL in human contexts offers significant advantages, such as reduced switching costs and enhanced specialization. However, it may also lead to drawbacks like increased physical or mental stress and alienation among workers.

Insect Societies:

In insect colonies, DoL manifests as:

- **Temporal Polyethism:** Age-related task allocation.
- **Morphological Castes:** Physical differentiation into roles (e.g., minors for routine tasks, majors for defense and heavy work).
- **Behavioral Castes:** Variations in individual response thresholds that lead to task specialization.

The plasticity of these systems allows insects to reallocate workers as environmental conditions change, a concept illustrated by Wilson's experiments with *Pheidole* ants.

4.3 Response Threshold Models

The chapter introduces response threshold models to explain task allocation:

- **Stimulus and Threshold:** Each task has an associated stimulus (s) and each individual a response threshold (θ). An individual engages in a task when the stimulus exceeds its threshold.
- **Mathematical Description:** A common formulation is

$$T_\theta(s) = \frac{s^n}{s^n + \theta^n},$$

  where the exponent n adjusts the sensitivity of the response. Alternative models, such as exponential functions, are also discussed.
- **Adaptation and Specialization:** Repeated task execution can lead to threshold adaptation, resulting in specialization and improved performance over time.
- **Stigmergy:** Interactions among individuals are mediated through environmental changes; for example, as one agent performs a task and reduces the stimulus, its neighbors are less likely to engage in that same task.

4.4 Robotic Implementations and Applications

Group Foraging:

Robotic experiments, such as those involving Khepera robots, demonstrate self-organized task allocation:

- Robots monitor a stigmergic variable—such as a nest's energy level—and decide whether to forage based on predefined thresholds.
- This decentralized mechanism mimics natural foraging behavior observed in social insects.

Adaptive Postmen Allocation:

The chapter also examines an urban application where mailmen are assigned to different zones:

- Each mailman has a threshold for responding to the demand in a city zone.
- As demand (the stigmergic variable) changes, thresholds adapt, ensuring that regions with higher needs receive more attention.

Interference Reduction and Task Partitioning:

Another key application is the reduction of interference in densely populated robotic swarms:

- Concepts such as the bucket brigade are used to illustrate how partitioning tasks into subtasks (for example, from source to an intermediate cache and then to a nest) can mitigate congestion.
- Experimental setups with e-puck robots show that dynamic task partitioning leads to enhanced efficiency by reducing waiting times and collisions.

4.5 Conclusion

The chapter concludes that the threshold model provides a powerful framework for understanding and designing DoL in both natural and artificial systems. By enabling dynamic task allocation and reducing interference, these principles facilitate resilient, adaptive, and scalable multi-agent systems. The inherent plasticity, allowing for continuous threshold adaptation and specialization, is essential for coping with fluctuating environmental demands.

5 From AutoMoDe to the Demiurge [5]

5.1 Motivation and Overview

Chapter 5 ([5]) explores the concept of *automatic design of robot swarms*, focusing on the AutoMoDe framework developed at IRIDIA. The main goal is to devise a systematic method for designing control software (and, eventually, hardware) for large-scale robot swarms. This approach stands in contrast to traditional hand-designed solutions or purely bio-inspired methods (e.g., classical evolutionary robotics). The chapter culminates with the vision of the *Demiurge*, an integrated system that automatically designs a swarm—from modules of control software to modules of hardware—based on user-specified requirements.

5.2 Introductory Concepts in Automatic Design

- **Self-Organization & Swarm Features:** Robot swarms are autonomous, fault-tolerant, scalable, and flexible, but lack a general engineering methodology for consistent design.
- **Classical Evolutionary Robotics (ER):** Often uses neural networks to map sensors to actuators, trained by evolutionary algorithms. However, from an engineering standpoint, classical ER may suffer from:
  - Excessive *variance* (overfitting to simulation idiosyncrasies).
  - Ambiguous experimenter intervention and limited real-world reliability (the *reality gap* problem).
- **Offline vs. Online Automatic Design:**
  - *Offline Design:* A single or iterative optimization phase in simulation. Once validated, the swarm controller is deployed in the real environment.
  - *Online Design:* Continuous adaptation in the target environment during operation (beyond the scope of the main examples here).
- **Reality Gap as Generalization:** The notion of bridging the reality gap is treated like a machine learning generalization problem. If a control solution overfits the simulator, it will perform poorly on physical robots.

5.3 AutoMoDe: A Bias-Driven Approach

Motivation.

To reduce overfitting and improve reliability, AutoMoDe injects *domain bias* into the controller design. Instead of letting an evolutionary algorithm freely evolve a high-dimensional neural network, AutoMoDe:

- Relies on *pre-defined, parametric behaviors* (*modules*) that are mission-independent but chosen to reflect typical robot capabilities (e.g., `exploration`, `phototaxis`, `repulsion`).
- Assembles them into *finite state machines* (or behavior trees) with *transition conditions* (also from a pre-defined set).

5.3.1 AutoMoDe-Vanilla

- **Reference Model (RM1):** Defines which robot sensors (proximity, light, ground, range-and-bearing, etc.) and actuators (wheels) are accessible. Control operates in 100 ms cycles.
- **Behaviors and Conditions:**
  - **Behaviors** include `exploration`, `stop`, `phototaxis`, `anti-phototaxis`, `attraction`, `repulsion`, etc. Each has a small set of numeric parameters.
  - **Conditions** include `black-floor`, `gray-floor`, `neighbor-count`, `fixed-probabili` and others. These conditions determine transitions between behaviors in the finite state machine.
- **Optimization Method (F-Race):** An iterative racing procedure that evaluates candidate designs in simulation and discards poorly performing ones. This process uses a budget of simulation runs (e.g., 10k, 50k, or 200k simulation steps) before converging to a final controller.

Experimental Missions and Formulas.

AutoMoDe-Vanilla was compared to a classical ER approach (*EvoStick*) on two standard missions:

1. **Aggregation:** Robots must gather around one of two circular areas.

$$F_{\text{aggregation}} = \frac{\max(N_a,\, N_b)}{N}$$

   where N_a and N_b are the counts of robots on each circular area, and N is the total number of robots.

2. **Foraging:** Robots collect objects and return them to a nest region.

$$F_{\text{foraging}} = N_0$$

where N_0 is the total number of items retrieved.

Simulations showed *EvoStick* controllers had higher performance in simulation, but in reality, they performed significantly worse than AutoMoDe, illustrating the reality gap. AutoMoDe's simpler, more biased approach generalized better.

5.3.2 Human vs. AutoMoDe-Vanilla Experiments

- **Five Missions Proposed by Human Designers:**
  1. **Shelter with Constrained Access (SCA)**: Robots aggregate in a *white shelter*, accessible only through a gap.

     $$F_{\text{SCA}} = \sum_{t=1}^{T} N(t)$$

     where $N(t)$ is the number of robots in the shelter at time t.
  2. **Largest Covering Network (LCN)**: Robots maximize coverage while remaining connected (via local range-and-bearing).

     $$F_{\text{LCN}} = A_C(T)$$

     an area-related metric capturing how widely they spread while preserving connectivity.
  3. **Coverage with Forbidden Areas (CFA)**: Robots move around to cover a region but must avoid black zones.

     $$F_{\text{CFA}} = \mathbb{E}\big[d(T)\big]$$

     which reflects the distribution or "distance" from forbidden areas over time.
  4. **Surface and Perimeter Coverage (SPC)**: Robots must cover a white square's surface *and* trace the perimeter of a black circle.

     $$F_{\text{SPC}} = \frac{\mathbb{E}\big[d_a(T)\big]}{c_a} + \frac{\mathbb{E}\big[d_p(T)\big]}{c_p}$$

     where $d_a(T)$ and $d_p(T)$ measure how well they cover the area/perimeter, and c_a, c_p are normalization constants.

5. **Aggregation with Ambient Cues (AAC)**: Robots aggregate in a designated black region.

$$F_{\text{AAC}} = \sum_{t=1}^{T} N_b(t)$$

where $N_b(t)$ is the number of robots on the black area at time t.

- **Design Methods:**
  - *Vanilla* and *EvoStick* (automatic).
  - *C-Human* (constrained human, who uses only the same pre-defined modules).
  - *U-Human* (unconstrained human, who writes C++ code freely).
- **Results:** On real robots, *Vanilla* consistently outperformed *EvoStick*, validated across all five missions. C-Human often performed best, but interestingly, U-Human sometimes faced the same reality-gap pitfalls as EvoStick, indicating that unconstrained freedom can lead to overfitting even for humans.

5.3.3 AutoMoDe-Chocolate

- Uses the *same* set of modules as Vanilla, but employs **iterated F-Race**, which refines candidate solutions by repeatedly sampling around the best FSM designs.
- Experiments again compared *Chocolate* against *Vanilla* and C-Human on the five missions.
- Chocolate typically outperformed *Vanilla* and, in many cases, approached or exceeded the *C-Human* results in real-robot tests. This underscores the value of stronger (yet still offline) optimization algorithms.

5.4 Beyond Vanilla and Chocolate

Other AutoMoDe Flavors.

Several new variants—*Gianduja*, *Waffle*, *Maple*, *Mate*, *TuttiFrutti*, *Coconut*, *IcePop*—extend the design approach to:

- Incorporate explicit robot communication (e.g., LED color signals).
- Integrate behavior-tree architectures.
- Employ different global optimization strategies (e.g., simulated annealing).
- Modify hardware assumptions (sensors, actuators) to increase or shift the design space.

5.5 Towards the Demiurge

Concept.

Demiurge is an envisioned integrated system that:

- Accepts **mission requirements** in a formal specification language.
- Automatically composes *both* hardware and software modules into a functional robot swarm design.
- Aims to handle the entire design pipeline with minimal or no human intervention.

By building on the successes of AutoMoDe, *Demiurge* will push towards a fully automated engineering approach where reusability of modules (and theoretical performance guarantees) become feasible.

5.6 Conclusions and Future Directions

- AutoMoDe's approach, grounded in machine learning, helps *mitigate the reality gap* through carefully chosen biases (finite-state modules).
- Empirical evidence shows that *injecting domain knowledge* (modules) plus robust *optimization* (F-Race, iterated F-Race) yields controllers that transfer more reliably from simulation to physical robots, often surpassing unconstrained evolutionary methods or naive human-coded solutions.
- The natural next step is the *Demiurge*, wherein *both* the hardware platform and the software architecture can be automatically orchestrated to meet user-specified mission requirements.

6 Security Issues in Swarm Robotics [6]

6.1 Motivation and Overview

This chapter tackles the growing concern of *security* in swarm robotics, concentrating on how malicious or faulty robots (often termed *Byzantine robots*) can threaten a swarm's performance. The lecture emphasizes:

- The difficulty of maintaining correct group decisions in a decentralized setting.
- The vulnerabilities introduced by adversarial robots that can falsify data, spoof identities (*Sybil attacks*), or deviate from intended behaviors.
- Potential solutions based on *blockchain technology* and *smart contracts*, which bring tamper-resistant logs, identity control, and secure coordination.

6.2 Part I: Shortcomings of Classical Approaches

Byzantine Robots and Consensus.

A central challenge in swarm robotics is **approximate consensus**, wherein robots must share sensor readings or partial decisions to converge on a correct global estimate. Traditional *Byzantine linear approximate consensus* algorithms (e.g., local averaging with outlier removal) can be severely compromised by:

- **Persistent false data**: A single robot injecting a constant incorrect reading (e.g., "0% white tiles") skews the swarm's final consensus.
- **Sybil Attacks**: A single physical unit can create many fake IDs, effectively weighting its incorrect reading many times. This enables a single adversary to overshadow the legitimate majority.

Empirical results show that even one strong Byzantine agent can dominate classical consensus, causing large errors or infinite convergence times if the swarm attempts to reject outliers in vain.

Implications.

Because swarm robots tend to rely on redundancy, local broadcasts, and no centralized oversight, once a malicious unit tampers with the communication protocol or spoofs IDs, the swarm may fail to converge or converge incorrectly. This underscores the need for additional security layers beyond naive averaging schemes.

6.3 Part II: Blockchain Technology Fundamentals

Peer-to-Peer Databases.

A *blockchain* is a replicated, tamper-resistant ledger maintained by a peer-to-peer (P2P) network. Key mechanisms include:

- **Cryptographic Hashing**: Each block references the hash of the previous block, so modifying any content invalidates subsequent blocks.
- **Proof of Work (PoW)** or other *consensus* protocols: Miners solve expensive puzzles, ensuring that rewriting history requires enormous computational effort.
- **Digital Signatures**: Transactions or data entries are verified with public-key signatures, confirming *who* submitted a piece of data.

Blockchain 1.0 vs. 2.0.

- **Blockchain 1.0 (Bitcoin)**: Focuses on decentralized digital currency (no need for a central bank).
- **Blockchain 2.0 (Ethereum)**: Generalizes the concept via *smart contracts*, i.e. code residing on the blockchain that executes deterministically on every node in the network. This allows arbitrary decentralized applications without central authorities.

6.4 Part III: Blockchain-Based Coordination of Robot Swarms

Smart Contracts for Security.

By storing *control logic* (a finite-state machine, or a consensus routine) in a **smart contract**, the swarm can:

- **Enforce** consistent updates: Each robot's data or vote is appended as a blockchain transaction, making it *immutable*.
- **Penalize** faulty behavior: If readings are contradictory, the contract can isolate or de-fund the malicious actor.

- **Prevent Sybil attacks**: If each robot must own a scarce cryptographic token or stake, it is costly/impossible to spawn fake identities at scale.

Illustrative Results.

Several experiments (including physical swarms of 20+ robots) demonstrate that running a consensus protocol via a blockchain contract can:

- **Preserve accuracy** even with multiple Byzantine robots.
- **Detect** or neutralize malicious agents by analyzing transaction patterns on the ledger.
- **Scale** to tens or hundreds of robots, storing minimal amounts of data ($\sim$160 bytes per message) and staying within feasible CPU/RAM budgets on low-power devices.

Economics of Tokens.

In some designs, *tokens* or *crypto-currency* held by each robot allow:

- **Self-Governance**: A newly joining robot must pay a membership stake, preventing quick creation of Sybil IDs.
- **Incentive Mechanisms**: Robots that abide by correct behaviors can earn tokens; malicious ones lose them, eventually losing influence in the swarm.

6.5 Advanced Directions

Distributed Learning.

A major open challenge is **secure machine learning** in swarms:

- *Federated Learning* or model aggregation can be used to train a shared model from distributed data, but *poisoning attacks* by a single Byzantine robot can degrade training.
- A blockchain-based contract can **detect outliers** or suspicious local updates, removing them from model aggregation.

Formalizing Compliance and Accountability.

Using smart contracts to record each robot's actions, or suspect readings, can yield **compliance with regulations** (e.g. for auditing whether a robot swarm was intentionally manipulated). Future directions include:

- **DAO (Decentralized Autonomous Organization)**: let robot-human collectives vote on high-level decisions (e.g., approving new swarm members or switching tasks).
- **Legal frameworks**: tying blockchain logs to real-world accountability if an owner's robot acts maliciously.

6.6 Conclusion

Chapter 6 ([6]) illustrates that while swarm robotics excels in flexibility and fault tolerance, *malicious or misbehaving* robots pose critical threats. Classical approximate consensus can quickly fail under Sybil or Byzantine attacks. By weaving **blockchain technology** and **smart contracts** into the swarm's communication and decision-making loops:

- We achieve *secure, tamper-evident* logs of each robot's state,
- We can *punish* or *exclude* malicious nodes,
- And we can build *novel* functionalities (secure machine learning, token-based economies, legal compliance) that harness the decentralized trust guaranteed by blockchains.

Continued research aims to refine these methods, ensuring that large-scale autonomous swarms remain robust, trustworthy, and auditably secure in real-world deployments.

References

[1] M. Dorigo, “1st lesson info-h-414 swarm intelligence - overheads,” Slides, Université Libre de Bruxelles (ULB), http://cs.ulb.ac.be/public/teaching/infoh414.

[2] ——, “2nd lesson info-h-414 swarm intelligence - overheads,” Slides, Université Libre de Bruxelles (ULB), http://cs.ulb.ac.be/public/teaching/infoh414.

[3] ——, “3rd lesson info-h-414 swarm intelligence - overheads,” Slides, Université Libre de Bruxelles (ULB), http://cs.ulb.ac.be/public/teaching/infoh414.

[4] M. Birattari, “4th lesson info-h-414 swarm intelligence - overheads: Division of labor and the threshold model,” Slides, Université Libre de Bruxelles (ULB), http://cs.ulb.ac.be/public/teaching/infoh414.

[5] ——, “5th lesson info-h-414 swarm intelligence - overheads: From automode to the demiurge,” Slides, Université Libre de Bruxelles (ULB), http://cs.ulb.ac.be/public/teaching/infoh414.

[6] V. Strobel, “6th lesson info-h-414 swarm intelligence - overheads: Security issues in swarm robotics,” Slides, Université Libre de Bruxelles (ULB), presented on 16 May 2024. http://cs.ulb.ac.be/public/teaching/infoh414.