University of Bamberg
Applied Computer Science

Seminar AI: yesterday, today, tomorrow

# Ant Colony Optimization

## An exploration of application criteria the context of pathfinding scenarios

by

Eva Klein

27th February 2017

advised by
Prof. Dr. Ute Schmid

# Contents

In our current day and age, we seem to be as keen as ever to optimize every tiny aspect of our lives. How to minimize the amount of water, electricity and calories we consume, how to maximize the profit of our enterprises or how to find the fastest, safest or most beautiful way from one location to another. This paper will present the Ant Colony Optimization Meta-Heuristic, a class of algorithms inspired by the ability of some ant species to find the shortest path to a food source with only local knowledge, and its applicability to three optimization tasks in the fields of rescue planning and robotic pathfinding.

# 1 Introduction: Algorithms inspired by Nature

The fascination of humankind with nature's ability to adapt to harsh, continuously changing environments has existed ever since Darwin's "The Evolutionof Species" made its impact into our societies and thought processes. Every day we can read about new accomplishments in science and technology, achieved by taking a closer look at mechanisms in our surrounding environment and imitating them. There are drones copying the flight behaviour of dragon flies, wall and car paint with "lotus effect", causing liquids to run off these surfaces in droplets. There are even surgical staples mimicking porcupine quills, creating smaller punctures and thus decreasing infection risks compared to common staples (Cho et al., 2012). But these advances are not only found in bioengineering, medicine or aerodynamics, they have entered the field of Computational Intelligence with the early works of Ingo Rechenberg, George Friedman and Michael Conrad in the 1960s and 1970s and especially in the form of Genetic Algorithms in the work of John Holland (1975).

This class of algorithms has been increasingly popular with researchers in recent years, not only due to the fact that a sheer endless supply of inspirational examples seems to exist and ways to observe nature and its mechanics are improving, but also because most of the algorithms can be used as meta-heuristics[1], producing approximate solutions to highly complex optimization problems in little time with only few resources. Examples of the newer nature-inspired algorithms known today are firefly algorithms, cuckoo search, bat algorithms and flower pollination algorithms (Yang, 2014).

Figure 1 on page 3 presents an attempt at categorising these and many other nature-inspired algorithms by the domain they originate from: Brownlee (2011) distinguishes between Neural Algorithms, Immune Algorithms, Stochastic Algorithms, Evolutionary Algorithms, Physical Algorithms, Probabilistic Algorithms and Swarm Algorithms. It is important to point out that while most of these classes are modelled analogously after a phenomenon occurring in nature, Neural Algorithms are not derived by analogy but by metaphor, as they are not modelled after biological neurons and synapses. Stochastic Algorithms on the other hand are not inspired by a specific phenomenon but occur generally in nature.

[1] A meta-heuristic is a "fairly general computational technique [...] that (is) typically used to solve numerical and combinatorial optimization problems approximately in several iterations" (Kruse et al., 2016, p. 138)

This paper will present one member of the growing number of Swarm Algorithms: *Ant Colony Optimization* (ACO), first introduced by Marco Dorigo in 1996 (Dorigo et al., 1996). First, we will take a look at the natural analogy which inspired its invention. Second, the ACO algorithm is presented in detail. In the third section, two concrete path finding applications of ACO are described (Goodwin et al. (2015); Purian et al. (2013)), including some advantages and disadvantages of ACO, as well as a direct comparison to a genetic algorithm implementation (Purian et al., 2013). A short summary and outlook will conclude this paper.

# 2 Ant Colony Optimization - From natural inspiration to algorithm

## 2.1 The double bridge experiment by Goss et al. (1989)

In 1989, Goss et al. discovered in a laboratory experiment that the Argentine ant species *Iridomyrmex humilis* is able to select the shortest path from their nest to a food source with great reliability, even though each ant itself only has a very limited possibility to orientate in the environment. This notion of a group of single, unintelligent agents showing intelligent global behaviour by exchanging information between one another is also known as *swarm intelligence*.

The setting of the experiment was the following: Laboratory colonies of the *Iridomyrmex humilis* had access to a food source via crossing a bridge. This bridge was formed by connecting two identical modules, each consisting of two branches of different length. All paths were wide enough to allow ants to pass one another in both directions. At the beginning of each module, the ants had to decide which path to choose, as well as on the way back to the nest. Each fork junction branch was at 30°angle to avoid bias towards one of the paths and to not restrict the ants movements by making sharp turns.

The results were that five to ten minutes after starting the experiment by placing the two modules, the first ants had crossed it and had discovered the food source at the other end. They turned back to the nest to recruit other ants to carry the food and all ants appeared to choose their paths randomly. After some minutes however, this abruptly changed and almost all ants visibly prefer the shortest branch.

They achieve this remarkable feat by communicating with each other via pheromone trails left behind while walking back and forth between food source and nest. These ant species is naturally biased towards following the path with the highest amount of pheromone, indicating that many other members of the colony have already used it. This technique of making changes to the environmental circumstances which then act as a stimulus to further activity by other insects is called *stigmergy* ("stigmergy, n.", 2016). Figure 2 on page 4 shows how the disposal of pheromones influenced the choice of the ants over time, simplified to only one bridge module by Kruse et al. (2016).

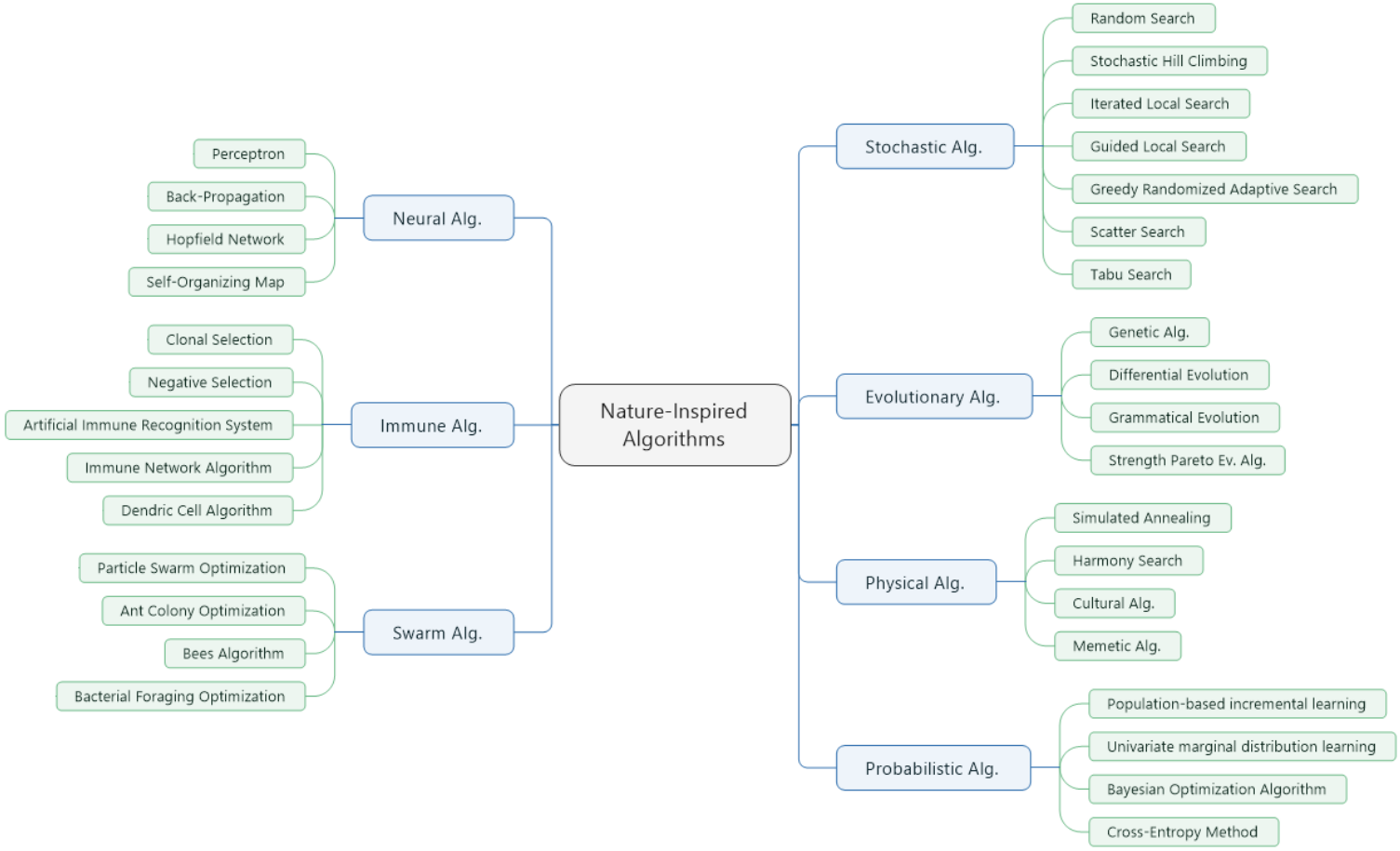


Figure 1: Overview of Algorithms inspired by nature, adapted from Brownlee (2011)

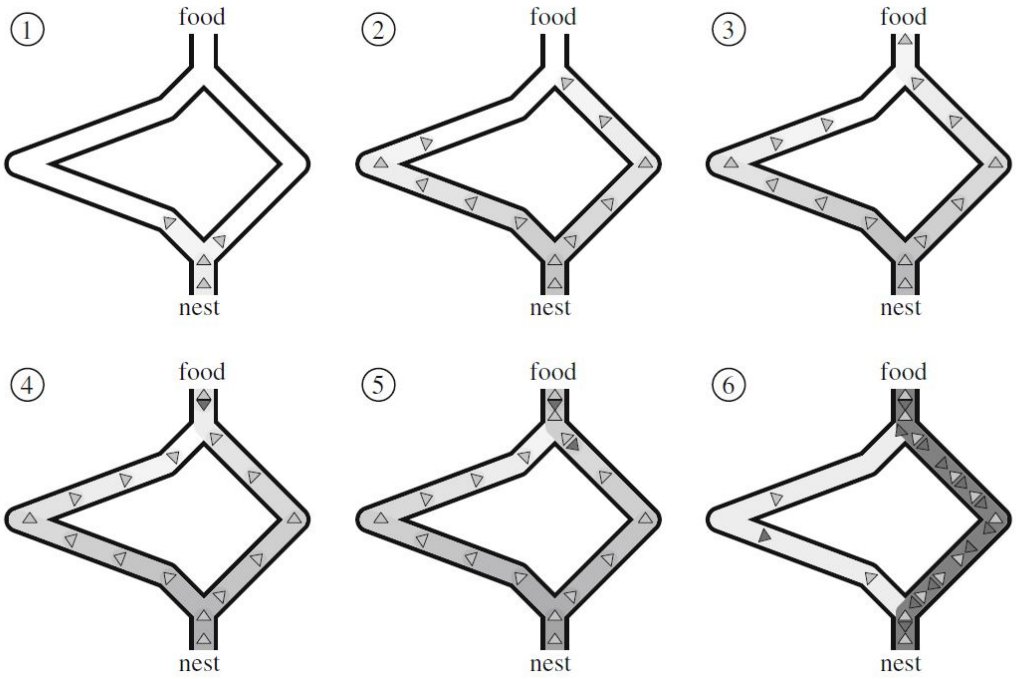


Figure 2: Double bridge experiment by Goss et al. (1989) as sketched by Kruse et al. (2016), p. 319. The shading indicates the amount of pheromone left on each path.

**1** No pheromone on both branches. Starting ants select branch randomly.

**2** About 50% of the ants choose the left branch and 50% the right one.

**3** As the right path is shorter, the ants on this branch reach the food source first.

**4** The ants from the right branch start to travel back to the nest. They have to choose a path to return and are biased by the higher pheromone level on the right branch. By doing so, they add another pheromone layer to the right path.

**5** More and more ants choose the right branch because of the increasing pheromone level.

**6** After some time the ants almost exclusively travel on the right path.

However, there is an undeniable disadvantage to the way ants determine the shortest path to a food source: In the event that an even shorter branch is added to the experimental set-up after the ants have already established a shortest trail, they will not recognize the new branch as being shorter but will continue on the previous path. This is due to the high bias towards the pheromone on the path, which hardly change over time.

We will see how this disadvantage is tackled by the ACO algorithm in section 2.2.2.

## 2.2 The Ant Colony Optimization Algorithm

To apply this knowledge about ants and their path-finding strategies to optimisation problems in the world of computational intelligence, certain steps are necessary.

First, the ants we are considering in this context are *artificial ants*, stochastic solution procedures which build candidate solutions. Each ant hereby represents a partial or complete candidate solution.

These artificial ants operate on a constructed graph, which can be a model of real-world locations, e.g. a building whose rooms are represented by vertices and the corridors and stairs as edges in the graph (cf. Goodwin et al. (2015)). But this graph representation also means that ACO can be applied to optimisation problems other than path finding, as will be shown in section 3.

Second, an additional heuristics $\eta$ can be introduced to improve edge selection and incorporate background knowledge, as we will see in section 2.2.1.

Third, most applications of ACO make use of a concept not commonly found in nature: pheromone cannot only build up on a path but they can also decrease over time. This *pheromone evaporation* is used to minimise the influence of old, less re-inforced pheromone to avoid premature convergence of the algorithm, as seen in the results of the double bridge experiment in section 2.1. It greatly improves the applicability of ACO in dynamic environments, as better candidate solutions are not ignored due to the fact that they were found in a later iteration than others.

With these adaptations in mind, we will now take a look at the typical structure of the ACO meta-heuristic:

**Algorithm 1** ACO Algorithm for combinatorial optimisation problems after Dorigo and Stützle (2010), p. 233

*Initialization*
**while** (termination condition not met) **do**
    Construct Ant Solutions
    Update Pheromones
**end while**

### Initialisation

In the beginning, all relevant parameters are set, for example the number of artificial ants $m$, a given heuristics function $\eta$, the initial pheromone level $\tau_0$ for each edge, the total amount $Q$ of available pheromone in each iteration and the the evaporation factor $\rho$.

### Termination

One possible termination criterion for Algorithm 1 is the moment when a candidate solution becomes "good enough" in the sense that it qualifies with respect to a pre-defined evaluation or fitness function, as it is the case in evolutionary algorithms. Secondly, the algorithm could terminate after a certain amount of time has passed and the algorithm

is then forced to terminate and return the best candidate solution found up to that time. Thirdly, a pre-set maximum of iteration could have been reached[2].

One iteration of algorithm 1 is completed after all ants have each constructed a candidate solution, i.e. reached the goal vertex.

### 2.2.1 Construction of Ant Solutions

In this step, each of the $m$ ants constructs a candidate solution to the problem setting at hand. They each start with an originally empty partial solution $s^p$ which is then expanded in each construction step by adding the next edge the ant chooses to walk on.

The key to the construction of ant solutions in algorithm 1 is the criterion by which an artificial ant selects this next edge in the graph. Formula 1 shows the most widely used selection rule of the original Ant System (Dorigo and Stützle, 2010, p. 234) and describes the probability $p_{i,k}$ with which an ant $k$ takes edge $e_{i,j}$ from vertix $i$ to vertix $j$:

$$p_{i,k}^k = \begin{cases} \frac{\tau_{i,j}^\alpha \eta_{i,j}^\beta}{\sum_{e_{i,l} \in N(s^p)} \tau_{i,l}^\alpha \eta_{i,l}^\beta} & \text{if } e_{i,j} \in N(s^p) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Where:

$$\begin{aligned} s &: \text{is a route from the source } v_s \text{ to the goal } v_e \\ s_p &: \text{is the partial solution of } s \\ \tau_{i,j} &: \text{is the number of pheromone on edge } e_{i,j} \\ N(s^p) &: \text{is the set of outgoing edges given } s^p \\ \eta_{i,j} &: \text{is the inverse heuristic estimate of the distance between vertices } i \text{ and } j \\ \alpha, \beta \in [0,1] &: \text{give weight to the pheromone } \tau \text{ and the heuristics function } \eta \end{aligned}$$

In the case that a partial solution cannot reach the goal state due to problem constraints, it is usually either discarded as infeasible or kept as a candidate solution and later penalized (Dorigo and Stützle, 2010), depending on the implemented construction mechanism.

### 2.2.2 Pheromone Update

After all ants have constructed a candidate solution, i.e. they have each found a route $s$ from the source $v_s$ to the sink node $v_e$, the pheromone on the graph edges is updated. Unlike in nature, where an ant constantly leaves pheromone behind while walking, most updating strategies add pheromone to the edges an ant used on its path to the sink only after that ant has reached the goal vertex, thus omitting edges which are not part of

[2]The second and third termination options are only possible because ACO falls into the category of so-called *anytime algorithms*, which means that it is able to return at least some available solution candidates at any point in time. It is important to note that most of the time, longer allocated CPU times are related to improving values (cf. Battiti and Brunato, 2010, p. 554).

successful searches. This method also tackles the problem of self-intensifying circles, as cycles can be removed from the path when then ant reaches $v_e$.

The two main mechanisms used in the updating step are firstly the depositing of pheromone to increase the pheromone level of those edges which are part of a (good) candidate solution and secondly the evaporation of pheromone, which decreases the pheromone deposited by ants in previous iterations.

Since the invention of the ACO in its original form, the *Ant System* by Dorigo et al. (1996), researchers have developed numerous extensions of this algorithm, which often differ in their pheromone distribution step. An overview of the most common updating strategies can be seen in Table 1 on page 8. This paper will focus on the original pheromone update implementation of *Ant System* as it is used by Goodwin et al. (2015) (see equation (2)).

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}. \tag{2}$$

Where:

$$\begin{aligned} m &: \text{is the total number of ants} \\ \rho &: \text{is the pheromone evaporation coefficient} \\ s &: \text{is a route from the source to the sink} \\ \Delta\tau_{i,j}^{k} &: \text{is the amount of pheromone deposited by the } k^{\text{th}} \text{ ant} \\ \Delta\tau_{i,j}^{k} &= \begin{cases} \frac{Q}{|s|} & \text{if } e_{i,j} \in s; Q \textit{ const.} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This strategy guarantees that only those edges receive new pheromone that are part of a candidate solution in the current iteration.

As Dorigo and Gambardella (1997b) point out, by allocating a greater amount of pheromone to shorter routes, the practice of pheromone update is similar to a reinforcement learning scheme. It comprises the addition of new pheromone left on the trails to a candidate solution as well as the evaporation of pheromone on less-travelled paths. The pheromone left on the edges of the graph thus acts as a "distributed long-term memory"(Dorigo and Gambardella, 1997b, p. 55), which is stored not within the individual ant but in the environment of the graph.

## 3 Two Application Examples - ACO in path-finding scenarios

Currently, there are hundreds of successful applications of the ACO meta-heuristic (Dorigo and Stützle, 2010, p. 244f), following the first use in the context of the Travelling Salesman routing problem by Dorigo et al. (1996).

Generally, these applications can be divided into two classes: First, there are those dealing with *static NP*-hard combinatorial optimization problems, which often use local search algorithms between constructing ant solutions and the pheromone update step

| **Name** | **Principle** | **Time of Update** | **Updated edges** | **Advantage/ Disadvantage** |
|---|---|---|---|---|
| **Ant System** (Dorigo and Gambardella, 1997b) | proportional update by all ants launched in each iteration at the end of the iteration | end of each iteration | all edges used by at least one ant | $\ominus$ not competitive with algorithms tailored to specific *NP*-hard problems |
| **Elitist Strategy** (Dorigo et al., 1996) | reinforcement of best known solution | end of each iteration | all edges used by at least one ant + additional pheromone on all edges of currently best solution | $\oplus$ faster convergence |
| **Strict Elitist Strategy** | only the best candidate solution is kept for the next iteration | end of each iteration | only edges of best candidate solution of last iteration or globally best candidate solution | $\ominus$ risk of premature convergence in local optimum |
| **Rank based Ant System** (Bullnheimer et al., 1997) | candidate solutions are ranked and only the $k$ best are kept for next iteration | end of each iteration | only edges of $k$ best candidate solutions of last iteration and optionally of one globally best candidate solution | $\oplus$ faster convergence |
| **Max-Min** (Stützle and Hoos, 2000) | uses $\tau_{min}$ and $\tau_{max}$ boundaries for the probability of selecting an edge and initialises edges to $\tau_{max}$ | end of each iteration | only edges of best candidate solution of last iteration or globally best candidate solution | $\oplus$ better exploration of search space<br>$\ominus$ slower convergence |
| **Ant Colony System** (Dorigo and Gambardella, 1997a) | ants select next edge using *pseudo-random proportional rule*, introducing a parameter to favour exploration over exploitation | after every edge selection | only edges of best candidate solution of last iteration or overall best candidate solution | $\oplus$ better exploration of search space as ants remove pheromone from edges when choosing them |

Table 1: An overview of common approaches to pheromone updating (cf. Tumeo et al., 2008; Dorigo and Stützle, 2010; Kruse et al., 2016)

in order to improve their performance. Dorigo and Stützle (2004) gives an overview over many such applications, which comprise problems from the fields of assignment, scheduling (e.g. car sequencing by Solnon (2008)), subset problems, machine learning (using so called *hybrid*-approaches which build on other existing classifiers like Decision Trees or SVMs) and Bio-Informatics (e.g. DNA Sequencing by Blum et al. (2008)).

The second class of applications are *dynamic* network routing problems, which are characterized by the circumstance that the availability of edges in the problem graph (here: links in the network) and the costs of traversing these edges are not constant, but time-dependent.

This chapter will explore two applications of ACO to path finding problems in dynamic environments comparable to those of network routing, namely the simulations of Goodwin et al. (2015) and Brand et al. (2010). We will see which aspects of the ACO meta-heuristic make it especially applicable in these two cases and finally how ACO performs compared to a genetic algorithm in Purian et al. (2013).

## 3.1 Comparison of application settings

ACO can be applied to any optimization problem that can be formulated as a search in a graph. A solution candidate must then be describable as a set of edges, which do not necessarily have to form a path, as long as there is a mechanism to iteratively select edges from the set. Each such selection can be seen as a decision about which edge will be added next to a partial candidate solution, and the construction of these candidate solutions is therefore equivalent to a sequence of decisions, which can be represented as a path in a decision graph.

In the examples we are going to explore in this chapter, however, the problems ACO is applied to are typical, but dynamic path finding problems: Goodwin et al. (2015) set out to propose a near-optimal escape plan for every person in a fire evacuation scenario, taking into account static environments, dynamic spread of the fire, movability and visibility impairments as well as incompleteness of information. Brand et al. (2010) use ACO to find the shortest, collision-free path - if one exists - between the starting point of a robot and a destination point in a grid network. To simulate dynamism, they added obstacles once one optimal path had been found.

The first step to apply ACO to any Optimization Problem is to translate it into a search graph. Table 2 shows how the two applications defined the nodes and edges of this graph, how they modelled contraints, how pheromone updates were handled, which heuristics were used (if any) and finally how the candidate solutions were generated.

It is important to point out that ACO algorithms typically achieve especially high performance values if the parameters are fine-tuned to the specific problem at hand, exploiting available background knowledge in form of heuristics, using fine-tuned local search algorithms or making informed choices for the construction mechanism (Dorigo and Stützle, 2010).

While both applications considered in this chapter do not openly use any heuristic function to influence edge selection, they both use background information to decide which edges receive more pheromone than others after each iteration.

| | **Goodwin et al. (2015)** | **Brand et al. (2010)** |
|---|---|---|
| **Application Domain** | Fire Escape Planning | Robot Path Planning |
| **Graph modelling** | Vertices: rooms of a ship or building<br>Edges: corridors | Grid of 20x20, 30x30 and 40x40 evenly distributed vertices. Inner edges only connect in the direction up, down, left and right. Robot: top-left, goal: bottom-right. |
| **Constraint handling** | $h(v_i, t)$ describes probability that there is a hazard at vertex $v_i$ at time $t$ (0=hazard-free)<br>$m(v_i, t)$ describes whether evacuee can move from $v_i$ | Barriers are added to the grid. The robot cannot move to vertices covered by these barriers. |
| **Dynamic Environment** | 5 stages represented via different hazard functions<br>1. random static assignment of $h(v_i) \in [0, 1]$<br>2. $h(v_i, t)$ switches to polar opposites every $n$ time units<br>3. $h(v_i, t)$ represents realistic probabilities based on exposure levels of heat radiation and temperature (computer simulation)<br>4. $m(v_i, t) \in true, false$ indicates whether evacuee can move away from room $v_i$ due to smoke obscuration<br>5. environment is updated with imperfect knowledge: only hazard functions close to evacuee are available (ad-hoc network setting) | Barriers which are proportional to the grid size and of different shapes and sizes are added after the algorithm converges once. Then the pheromone is re-initialized comparing two variants:<br><br>Global initialization: $\tau$ is reset to 0.1 for all edges.<br><br>Local initialization: $\tau_{i,j}$ is set to 50% of the highest pheromone level of the last iteration if $e_i, j$ is directly next to the barrier. With growing distance from the barrier, $\tau_{i,j}$ decreases by a pre-defined fraction. |
| **Pheromone update** | See (2). $Q$ replaced by $Q(s,t) = \prod_{v_i \in s}(1 - h(v_i, t))$ (inv. hazard probability on route $s$ at time $t$),$\rho = 0.2$ | See (2). $\Delta\tau^k_{i,j} = \frac{1}{C^k}$ where $C^k$ is the cost or reward of ant $k$ choosing $e_{i,j}$, here: total path length |
| **Heuristics** | – | – |
| **Solution generation** | See (1) with $\beta = 0$ | See (1), $\tau_0$ is initialized to 0.1 on all edges |

Table 2: A comparison of the simulation settings in Goodwin et al. (2015) and Brand et al. (2010)

Goodwin et al. (2015) use so-called hazard functions to determine whether an edge belongs to a safe path or to one containing risks and include this information in the form of $Q(s,t) = \prod_{v_i \in s}(1 - h(v_i, t))$ in the pheromone update step.

Brand et al. (2010) on the other hand use the technique of re-initializing those edges which are close to a newly added barrier even before the ants started their exploration, therefore heavily drawing from global knowledge.

There is therefore a noticeable difference between the two applications. While both use a considerable amount of background knowledge to increase their performance, Goodwin et al. (2015) clearly state where this knowledge originates: In their experimental setting the realistic radiation and smoke development included in the hazard functions was provided by the Fire Dynamics Simulator tool, but in a realistic mobile-app implementation, this information would come from fixed or mobile sensors. In contrast, the information source about the location of the obstacles in Brand et al. (2010) is not mentioned, leaving some open questions.

## 3.2 Results

Both approaches show that ACO can be very effective, especially in dynamic environments. Goodwin et al. (2015) conclude that in all 5 scenarios tested (see Table 2), ACO was empirically able to reach a near-optimal solution as long as the evaporation coefficient $\rho$ was not set to 0. They especially showed what (Dorigo and Stützle, 2010, p.254) prophesied as the "greatest advantage" of ACO algorithms: An application in a highly dynamic domain where only local information is available. The last experimental scenario worked only with hazard functions containing sensor data from 5, 10, 25 or 50 rooms around the current location of an ant. While solutions had to be updated again and again after gaining new information with each changing room, even with only data from 5 rooms in each direction 100 ants were able to keep the probability of an evacuee encountering a hazard on the calculated route below 2% for almost 10 minutes after the fire started.

Brand et al. (2010) showed that their mechanism of a global initialization after the addition of an obstacle needed 23.8% more iterations to find an optimal path than the localized initialization. This further confirms the assumption that those extensions of ACO which can include more background knowledge and which are more closely tailored to the problem at hand out-perform their counterparts.

All in all, both papers show that "[ACO] techniques tend to exhibit a high degree of flexibility and robustness in a dynamic environment"(Bonabeau et al., 2000, p. 39). The advantage of an on-line application allowing for dynamic re-routing if obstacles appear or environment grows hostile is apparent.

## 3.3 Comparison to Genetic Algorithm

As the applications considered in sections 3.1 and 3.2 do not provide any comparison to other algorithms used in dynamic environments, this chapter will present a short comparison of the key characteristics of ACO and Genetic Algorithms as well as a comparison

of performance values from Purian et al. (2013).

### 3.3.1 Characteristics

While many sources (cf. Brownlee, 2011; Kruse et al., 2016; Gerdes et al., 2004) do not consider ACO to be a sub-class of Genetic or Evolutionary Algorithms, but of Swarm Intelligence Algorithms (recall Figure 1), there are some noticeable similarities.

Both are algorithms inspired by nature and fall into the category of *population-based* algorithms. In genetic algorithms, a population is a multiset of candidate solutions which are subject to the selection process and ranked via a fitness function (Kruse et al., 2016, p. 193f). These populations correspond to the candidate solutions constructed by artificial ants in each iteration of the ACO.

Also, both algorithms do not guarantee to return a globally optimal solution but are *anytime* algorithms that find the best solution up to the point where the algorithms is stopped.

But where Genetic Algorithms construct new candidate solutions in each iteration by certain mutation and selection steps, ACO algorithms do not evolve in their genetics but through communication with the environment. Each new iteration (or "generation") of ants does not differ from the previous generation, but the environment has been changed by the help of artificial pheromone. Thus, ACO algorithms can be said to evolve through their social behaviour (cf. Elbeltagi et al., 2005, p. 46).

When it comes to mutation, Yang (2014) notes that ACO's step is "not as simple as flipping digits" but that new solutions are generally constructed by fitness-proportional mutation, meaning that only some of the candidate solutions receive additional pheromone, providing the basis for the next iteration. The author also points out, that unlike Genetic Algorithms, ACO algorithms lack an explicit crossover, which can lead to slower convergence in comparison [3].

### 3.3.2 Performance

Purian et al. (2013) provide a direct performance comparison of ACO and a Genetic Algorithm. Their simulation setting was as follows:

The genetic algorithm used a chromosome structure of variable length. Here each gene expresses the number of intermediate nodes along with the path determined by each chromosome. Thus the first gene represents the initial point and the last gene the target. First, a random initial population is generated by starting at the start node and then successively adding one of the 8 surrounding nodes which are not on the black list. If choosing a node decreases the angle to the target, this node has a higher probability of being chosen. The members of this first population are then ranked using their length and smoothness of their angles in each step. Following an elitist strategy, the 5 best chromosomes are then selected as the new parent generation. New solutions are generated by randomly selecting two parents and using a crossover with a probability

[3] Many extensions of the original *Ant System* counteract this deficit by introducing parameters to increase exploitation, e.g. *Ant Colony System* (see Table 1 on page 8)

| | **Purian et al. (2013)** |
|---|---|
| **Application Domain** | Robot Path Planning |
| **Graph modelling** | Networked environment of the moving robot divided into equally sized cells<br>This grid is populated by fixed and moving obstacles, forming three workspaces of increasing complexity<br>Vertices: grid cells, edges: connecting one cell with 8 surrounding neighbours (also diagonally) |
| **Constraint handling** | Static and moving obstacles. A "black list" of fixed and moving barriers is re-calculated for every time unit. If a node is part of the black list, the robot cannot move there. |
| **Dynamic Environment** | Moving obstacles with different speeds, sizes and directions |
| **Pheromone update** | Rank-based strategy: only 5 best candidate solutions release pheromones on their edges according to their ranking. $\rho = 0.1$ |
| **Heuristics** | Takes into account the angle between the next possible node and the target, this favouring the "right" direction |
| **Solution generation** | See (1) with $\beta = 0.5$, $\alpha = 1$, $\tau_0 = 0.5$ and $N(s^p)$ the adjacent nodes which are not on the "black list" |

Table 3: Simulation settings in Purian et al. (2013)

of 0.9 and a mutation step with probability 0.05 (for further details, see Purian et al. (2013, p. 33f)).

Even though both algorithms were designed in a similar way, using the same heuristics and an elitist selection strategy, the ACO implementation outperformed the Genetic Algorithm in all workspace settings modelling a routing environment for a moving root. It was able to find the optimal path using a smaller population (number of ants) and fewer iterations of the algorithm. While both algorithms needed approximately the same amount of time to finish one iteration, ACO needed significantly less time to find a collision-free route from start to target as it needed fewer iterations in general. The results of the simulations can be seen in Table 4.

| **Workspace complexity** | **low** | | **medium** | | **high** | |
|---|---|---|---|---|---|---|
| **Algorithm** | ACO | Genetic | ACO | Genetic | ACO | Genetic |
| **Time in** $s$ | 47.721 | 1522.03 | 684.69 | 1865.28 | 972.65 | 2560.42 |
| **Iterations** | 30 | 50 | 45 | 80 | 50 | 100 |
| **Population size** | 20 | 20 | 25 | 35 | 30 | 50 |

Table 4: Runtime results of ACO and Genetic Algorithm from Purian et al. (2013, p. 41)

The authors explained these different results by pointing out that as the complexity of the environment increases, the population size of the Genetic Algorithm needs to be adapted accordingly. ACO on the other hand does not need increase its population, as in each iteration only one ant alone will construct an entire candidate solution whose length is variable.

### 3.4 Possible disadvantages

After having shown some of the advantages in efficiency and robustness of ACO, there are also some disadvantages to keep in mind.

Especially in safety-critical environments such as fire rescue scenarios, one should keep in mind that ACO remains a meta-heuristic without any guarantee to find an optimal solution at a known point in time. These scenarios are built on the expectancy to find an optimal way from the fire to the evacuation zone, i.e. a path where the probability that a person encounters a hazard in at least one of the vertices in the chosen route is 0. Dorigo and Stützle (2004) proved that as the number of computation steps approach infinity, the probability that the optimal solution is found approaches 1. However, there is no guarantee as to when or if this optimal solution is reached with iterations below infinity, which is an insecurity factor hard to handle when every second counts and less optimal paths may lead to injury or death.

Another disadvantage of the ACO is that it can take quite long until the algorithm returns failure, i.e. that there is no path from the source to the sink in the graph. It can only report this non-existence with certainty after having visited all nodes in the graph, having thus constructed a partial candidate solution which can no longer be extended - and this for all ants in this iteration. In many applications where life and death do not depend on a few seconds or minutes, this might be acceptable, but especially in cases like Goodwin et al. (2015), any additional minute needed might see the fire spread to block another path to safety.

## 4 Summary and Outlook

Since its beginning, Ant Colony Optimization has seen many improvements and alterations to customize it to many different application scenarios. It is this flexibility of a relatively simple, resource-efficient meta-heuristic that still drives researchers to invent new variants and create hybrids with other optimization techniques to further explore where the limits of ACO truly lie. Current developments have been presented at the ANTS 2016, Tenth International Conference on Swarm Intelligence.

One of the most recent advances in this direction has been made by Goodwin and Yazidi (2016) who introduced an Ant Colony based classifier operating in two dimensional space, utilizing ray casting. This extended the applicability of ACO from stochastic optimization problems to classification, which had only been achieved by hybrid models before. Their results were promising, as the new system outperformed existing classifiers, such as polynomial and linear Support Vector Machines.

ACO's true powers lie in this possibility to extend it in many directions and the fact that it is explicitly formulated in terms of computational agents, making it generally very useful in the design of problem-solving systems (Bonabeau et al., 2000, p. 40). Used with the disadvantages of section 3.4 in mind, ACO may be a candidate for many other unsolved optimisation - and now even classification - problems.

In a time, where an algorithm inspired by ants can save lives in fire escape planning and even leads to the development of a mobile application to aid people in such dynamic situations using only incomplete knowledge (cf. Radianti et al., 2015), we can only imagine what other natural analogies still lie in the dark that can be used in their own creative and unique way in a computational intelligence setting.

## References


Roberto Battiti and Mauro Brunato. 2010. Reactive search optimization: learning while optimizing. In *Handbook of Metaheuristics*. Springer, 543–571.

Christian Blum, Mateu Yábar Vallès, and Maria J Blesa. 2008. An ant colony optimization algorithm for DNA sequencing by hybridization. *Computers & Operations Research* 35, 11 (2008), 3620–3635.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. 2000. Inspiration for optimization from social insect behaviour. *Nature* 406, 6791 (2000), 39–42.

Michael Brand, Michael Masuda, Nicole Wehner, and Xiao-Hua Yu. 2010. Ant Colony Optimization algorithm for robot path planning. In *International Conference on Computer Design and Applications (ICCDA), 2010*, Jinkuan Wang (Ed.). IEEE, Piscataway, NJ, V3–436–V3–440. DOI:http://dx.doi.org/10.1109/ICCDA.2010.5541300

J. Brownlee. 2011. *Clever Algorithms: Nature-inspired Programming Recipes*. Lulu.com. https://books.google.de/booksid=SESWXQphCUkC

Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. 1997. A new rank based version of the Ant System. A computational study. (1997).

Woo Kyung Cho, James A Ankrum, Dagang Guo, Shawn A Chester, Seung Yun Yang, Anurag Kashyap, Georgina A Campbell, Robert J Wood, Ram K Rijal, Rohit Karnik, and others. 2012. Microstructured barbs on the North American porcupine quill enable easy tissue penetration and difficult removal. *Proceedings of the National Academy of Sciences* 109, 52 (2012), 21289–21294.

Marco Dorigo and Luca Maria Gambardella. 1997a. Ant colonies for the travelling salesman problem. *biosystems* 43, 2 (1997), 73–81.

Marco Dorigo and Luca Maria Gambardella. 1997b. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation* 1, 1 (1997), 53–66.

Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. 1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26, 1 (1996), 29–41.

Marco Dorigo and Thomas Stützle. 2004. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA.

Marco Dorigo and Thomas Stützle. 2010. Ant Colony Optimization: Overview and Recent Advances. In *Handbook of Metaheuristics*, Michel Gendreau and Jean-Yves Potvin (Eds.). International Series in Operations Research & Management Science, Vol. 146. Springer Science + Business Media, s.l., 227–263. DOI:http://dx.doi.org/10.1007/978-1-4419-1665-5_8

Emad Elbeltagi, Tarek Hegazy, and Donald Grierson. 2005. Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics* 19, 1 (2005), 43–53. DOI:http://dx.doi.org/10.1016/j.aei.2005.01.004

Ingrid Gerdes, Frank Klawonn, and Rudolf Kruse. 2004. *Evolutionäre Algorithmen: Genetische Algorithmen – Strategien und Optimierungsverfahren – Beispielanwendungen.* Vieweg+Teubner Verlag, Wiesbaden.

Morten Goodwin, Ole-Christoffer Granmo, and Jaziar Radianti. 2015. Escape planning in realistic fire scenarios with Ant Colony Optimisation. *Applied Intelligence* 42, 1 (2015), 24–35. DOI:http://dx.doi.org/10.1007/s10489-014-0538-9

Morten Goodwin and Anis Yazidi. 2016. Ant Colony Optimisation-Based Classification Using Two-Dimensional Polygons. In *International Conference on Swarm Intelligence.* Springer, 53–64.

S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. 1989. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* 76, 12 (1989), 579–581. DOI: http://dx.doi.org/10.1007/BF00462870

John H Holland. 1975. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press* (1975).

Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. 2016. *Computational intelligence: a methodological introduction.* Springer.

Fatemeh Khosravi Purian, Fardad Farokhi, and Reza Sabbaghi Nadooshan. 2013. Comparing the performance of genetic algorithm and ant colony optimization algorithm for mobile robot path planning in the dynamic environments with different complexities. *Journal of Academic and Applied Studies* 3, 2 (2013), 29–44.

Jaziar Radianti, Mehdi Ben Lazreg, and Ole-Christoffer Granmo. 2015. Fire simulation-based adaptation of SmartRescue App for serious game: Design, setup and user experience. *Engineering Applications of Artificial Intelligence* 46 (2015), 312–325.

Christine Solnon. 2008. Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research* 191, 3 (2008), 1043–1055.

"stigmergy, n.". 2016. *OED Online.* Oxford University Press.

Thomas Stützle and Holger H Hoos. 2000. MAX–MIN ant system. *Future generation computer systems* 16, 8 (2000), 889–914.

Antonino Tumeo, Christian Pilato, Fabrizio Ferrandi, Donatella Sciuto, and Pier Luca Lanzi. 2008. Ant colony optimization for mapping and scheduling in heterogeneous

multiprocessor systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on*. IEEE, 142–149.

Xin-She Yang. 2014. *Nature-inspired optimization algorithms*. Elsevier.