Expert Systems with Applications 36 (2009) 9608–9617

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Review

A review of ant algorithms

R.J. Mullen *, D. Monekosso, S. Barman, P. Remagnino

Digital Image Research Centre, Kingston University, Penrhyn Road, London, England, UK

ARTICLE INFO

Keywords:
Ant algorithms
Swarm intelligence
Multi-agent systems
Machine learning

ABSTRACT

Ant algorithms are optimisation algorithms inspired by the foraging behaviour of real ants in the wild. Introduced in the early 1990s, ant algorithms aim at finding approximate solutions to optimisation problems through the use of artificial ants and their indirect communication via synthetic pheromones. The first ant algorithms and their development into the Ant Colony Optimisation (ACO) metaheuristic is described herein. An overview of past and present typical applications as well as more specialised and novel applications is given. The use of ant algorithms alongside more traditional machine learning techniques to produce robust, hybrid, optimisation algorithms is addressed, with a look towards future developments in this area of study.

1. Introduction

Optimisation problems can be found in many areas of industry, as well as in the scientific community. An example of an industry based optimisation problem is that of logistical traffic routing, where the goal may be simply to determine the most efficient route for a set of delivery vehicles to take. A more specialised and scientific based application exists in the computational biology community, where predicting the structure of proteins formed from their linear sequences is an important problem. In short, optimisation problems are wide ranging and plentiful; and as such, methods for solving these problems have been, and continue to be, a widely researched topic.

There exists a particular subset of optimisation problems, known as Combinatorial Optimisation (CO) problems (Papadimitriou & Steiglitz, 1982), in which a positive cost value is assigned to each object in the search space by an objective function, and the goal is to find an object of minimal cost.

Algorithms that guarantee the optimal solution to CO problems exist, but often at great computational cost, especially when dealing with NP-hard CO problems. For NP-hard CO problems, the computational time required to compute the optimal solution is often too high for any practical implementation. With this, approximation algorithms have received much attention, in order to compute accurate solutions in significantly less time.

Ant algorithms are one of the most recent approximate optimisation methods to be developed. These algorithms are inspired by the bahaviour of real ants in the wild (Dorigo, Maniezzo, & Colorni, 1996), and more specifically, by the indirect communication between ants within the colony via the secretion of chemical pheromones. Within the Artificial Intelligence (AI) community, ant algorithms are considered under the category of *swarm intelligence* (Bonabeau, Dorigo, & Theraulaz, 1999). Swarm intelligence encompasses the implementation of intelligent multi-agent systems that are based on the behaviour of real world insect swarms, as a problem solving tool. Other such algorithms that have been developed are based on, for example, the behaviour of swarms of wasps and bees. Continuing from the success of the original ant algorithm proposed by Dorigo et al. (1996), further development lead to a more general purpose optimisation technique known as Ant Colony Optimisation (ACO), which was later formalised into a metaheuristic[1] in (Dorigo & Di Caro, 1999; Dorigo, Di Caro, & Gambardella, 1999). Examples of other metaheuristics include simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), tabu search (Glover, 1989, 1990) and local search (Lourenco, Martin, & Stuzle, 2002).

The rest of this paper is organised as follows: Section 2 will introduce the ideas behind ant algorithms and where the biological inspiration comes from. Section 3 details how these ideas were put into practice as optimisation algorithms. Section 4 explains the consolidation into the ACO metaheuristic and its workings. Section 5 gives a brief performance comparison of some of the 'main' ACO algorithms. Section 6 looks at some of the more typical applications as solved by the 'main' ACO algorithms. Section 7 takes a look at more specialised and novel ant algorithms and their applications. Section 8 discusses where ant algorithms lie with respect to other machine learning techniques, and how ant algorithms can be used in conjunction with other machine learning

* Corresponding author.
E-mail address: r.mullen@kingston.ac.uk (R.J. Mullen).

[1] A *metaheuristic* is a set of guide lines for algorithmic development that can be applied to differing optimisation problems with little change necessary.

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.01.020

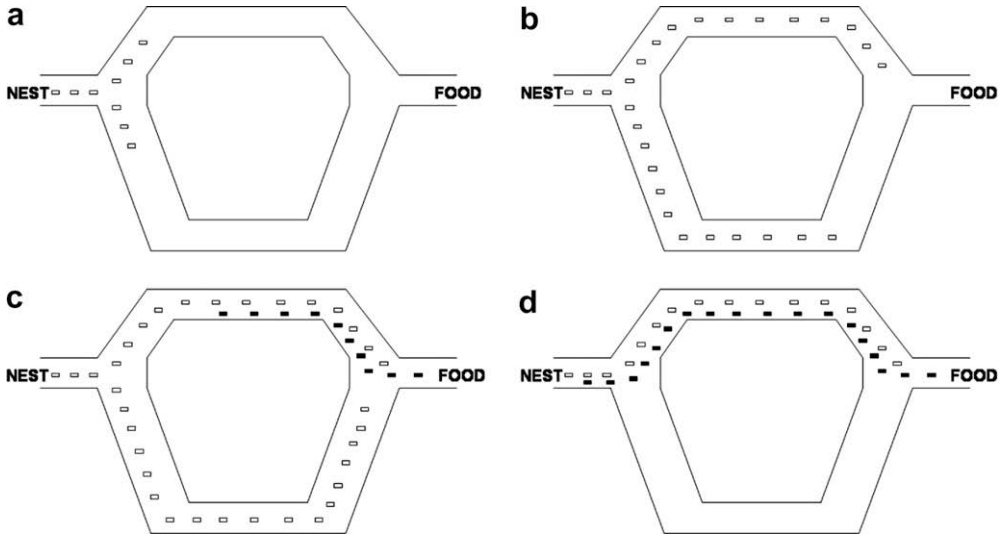

Fig. 1. A schematic illustration of the "Shortest bridge" experiment (Cordon et al., 2002).

techniques. Finally, Section 9 draws this paper to a close with concluding remarks.

2. Ants in the wild

The original inspired optimisation idea behind ant algorithms came from observations of the foraging behaviour of ants in the wild, and moreover, the phenomena known as *stigmergy*. A term introduced in 1959 by Grasse (1959), stigmergy refers to the indirect communication amongst a self-organising emergent system via individuals modifying their local environment. Experimental studies carried out by Deneubourg, Aron, Goss, and Pasteels (1990) explore the stigmergic nature of ant colonies, in which ants communicate indirectly by laying down pheromone trails, which ants then tend to follow. Experiments in Goss, Aron, Deneubourg, and Pasteels (1989) show the convergence of ant trails as a result of the tendency of ants to follow a trail that contains a higher concentration of pheromone deposit.

2.1. The analogy

Let us consider an experiment we shall call the "shortest bridge" experiment (Goss et al., 1989). Fig. 1 shows a diagram representing two possible paths between a nest and a food source, with the upper path shorter in length than that of the lower. Initially, as the ants leave the nest foraging for food, approximately half of the ants will follow the upper path and half the lower (Fig. 1a). Due to the upper path being shorter than the lower, the ants following the upper path will reach the food source sooner (Fig. 1b). Once an ant has collected food it returns to the nest by following the pheromone trail laid down on the way to the food source, again laying pheromone on the way, thus further reinforcing the pheromone trail along this path (Fig. 1c). These ants will return to the nest first, reinforcing the pheromone trail along the whole of the upper path. Any ants now leaving the nest will be more likely to follow the upper path from the nest to the food due to the higher pheromone concentration along this route, likewise for any ants returning to the nest from the food source. In this way eventually most, if not all, of the ants end up following the upper path and convergence to the shortest path is achieved (Fig. 1d). Essentially what we see with the foraging behaviour of ants in the wild can be described in a more computational way as an intelligent multi-agent system solution to a shortest-route optimisation problem, where in this case the agents in the system are ants. It is these observations that inspired the first ant algorithm (Dorigo et al., 1996), which was applied originally to the well known travelling salesman "benchmark" problem.

3. From nature to computers

The transition from the natural to artificial ant colony involves the use of simple computational agents that work cooperatively, communicating through artificial pheromone trails. In an iterative fashion,[2] each ant moves from state S_i to state S_j guided by two main factors:

1. *Heuristic information*: a measure of the heuristic preference (which is application based) for moving from state S_i to state S_j. This information is known *a priori* to the algorithm run, and is not modified during.
2. *Artificial pheromone trail(s)*: a measurement of the pheromone deposition from ants previous transitions from state S_i to state S_j. In other words a measurement of the "so far" learned preference. This information is modified during the algorithm run by the artificial ants.

3.1. Properties of the artificial ant

The following details the main properties associated with the artificial ant (Cordon, Herrera, & Stutzle, 2002):

- Each artificial ant has an internal memory which is used to store the path followed by the ant (i.e. the previously visited states).
- Starting in an initial state $S_{initial}$ each ant tries to build a feasible solution to the given problem, moving in an iterative fashion through its search space/environment.
- The guidance factors involved in an ants movement take the form of a transition rule which is applied before every move from state S_i to state S_j. The transition rule may also include

[2] Unlike natural ants, artificial ants exist in a discretised world; that is, they move from discrete state to discrete state.

additional problem specific constraints and may utilise the ants internal memory.
- The amount of pheromone each ant deposits is governed by a problem specific pheromone update rule.
- Ants may deposit pheromones associated with states, or alternatively, with state transitions.
- Pheromone deposition may occur at every state transition during the solution construction. This is known as *online step-by-step pheromone trial update*.
- Alternatively ants may retrace their paths once a solution has been constructed and only then deposit pheromone, all along their individual paths. This is known as *online delayed pheromone update*.

3.2. Additional characteristics

In addition to the above properties, artificial ants may also have characteristics to improve their performance that do not have a natural counterpart. Examples of widely used additional characteristics include *local search* (Lourenco et al., 2002; Dorigo & Gambardella, 1997) and *candidate list* (Dorigo & Di Caro, 1999; Dorigo & Gambardella, 1997).

Depending on the problem to be solved, daemon actions may be introduced into the algorithm. Daemon actions influence the guidance of the ants during algorithm runtime and can be used in order to speed up convergence. An example of a daemon action may be adding extra pheromone to the best solution trail so far, at the end of each iteration.

3.3. The original Ant System

The first ant algorithm, named "Ant System" (AS), was developed in the nineties by Dorigo et al. (1996). As a test-bed for this algorithm the well known benchmark Travelling Salesman Problem (TSP) was used.

For a set of M towns, the TSP problem involves finding the shortest length closed tour visiting each town only once. In other words finding the shortest-route to visit each town once, ending up back at the starting town. In the case of the Euclidean TSP we consider the path length between any given towns i and j to be the Euclidean distance between i and j, such that the path length $d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$.

Each ant in the system has the following characteristics:

- An ant decides which town to go to using a transition rule that is a function of the distance to the town and the amount of pheromone present along the connecting path.
- Transitions to already visited towns are added to a *tabu list* and are not allowed.
- Once a tour is complete, the ant lays a pheromone trail along each path visited in the tour.

An *iteration* is defined here as the interval in $(t, t+1)$ where each of the N ants moves once. We then define an *epoch* to be every n iterations, when each ant has completed a tour. After each epoch the pheromone intensity trails are updated according to the following formula:

$$\tau_{ij}(t+n) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{N} \Delta\tau_{ij}^{k}, \tag{1}$$

where $\rho \in (0,1]$ is the evaporation rate and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone laid on path (i,j) by the kth ant between time t and $t+n$, and is given by

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where Q is a constant and L_k is the tour length of the kth ant.

The heuristic information in this case is called the *visibility*, η_{ij}, and is defined as the quantity $1/d_{ij}$. As opposed to the pheromone trail, this quantity is not modified during the algorithm run. A *tabu list* is implemented as a growing vector containing the list of the previous and current visited towns. $tabu_k(s)$ gives the sth town visited by the kth ant in the current tour.

The probability of the kth ant making the transition from town i to town j is given by

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}]^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $allowed_k = \{M - tabu_k\}$ and α and β control the relative importance of the pheromone trail and visibility, respectively.

4. Ant algorithms as computational optimisation techniques

The success of AS as applied to the classic TSP test-bed led to the development of the ACO metaheuristic (Dorigo & Di Caro, 1999; Dorigo et al., 1999). The ACO metaheuristic was developed to describe, in a more general way, the overall method of solving combinatorial problems by approximate solutions based on the generic behaviour of natural ants. In this way, all specific applications of the ACO metaheuristic may be described as ant algorithms, however, ant algorithms are not confined to ACO.

4.1. The ACO metaheuristic

ACO is structured into three main functions (Algorithm 1). *AntSolutionsConstruct*( ) performs the solution construction process as described previously. Artificial ants move through adjacent states of a problem according to a transition rule, iteratively building solutions. *PheromoneUpdate*( ) performs pheromone trail updates. This may involve updating the pheromone trails once complete solutions have been built, or updating after each iteration.

In addition to pheromone trail reinforcement, ACO also includes pheromone trail evaporation. Evaporation of the pheromone trials is included to help ants 'forget' bad solutions that were learned early on in the algorithm run. Implementation could be as simple as reducing all pheromone trials by a set amount after each epoch. *DeamonActions*( ) is an optional step in the algorithm which involves applying additional updates from a global perspective (there exists no natural counterpart). An example could be applying additional pheromone reinforcement to the best solution generated (known as *offline pheromone trail update*).

Algorithm 1. The ant colony optimisation metaheuristic

```
ParameterInitialisation
WHILE termination conditions not met do
  ScheduleActivities
    AntSolutionsConstruct( )
    PheromoneUpdate( )
    DeamonActions( ) optional
  END ScheduleActivities
END WHILE
```

4.2. Developed ACO algorithms

Following from the original AS, various improvements were made which gave rise to several other ant algorithms which collectively form the main ACO algorithms; described below.

4.2.1. Max–Min Ant System

The Max–Min Ant System (MMAS) algorithm (Stutzle et al., 2000) differs from the AS in two main ways: only the best ant updates the pheromone trials, and the pheromone update function is bound. The modified pheromone update is as follows:

$$\tau_{ij}(t+n) = \left[(1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{N} \Delta\tau_{ij}^{best} \right]_{\tau_{min}}^{\tau_{max}}, \tag{4}$$

where

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & \text{if } (i,j) \text{ belongs to the best tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

and where τ_{max} and τ_{min} are the upper and lower pheromone bounds respectively. If we let the update function be x, the upper and lower bounds are imposed in the following way:

$$[x]_b^a = \begin{cases} a & \text{if } x > a, \\ b & \text{if } x < b, \\ x & \text{otherwise} \end{cases} \tag{6}$$

L_{best} is the length of the tour of the best ant. This can be either the *iteration-best*, the *best-so-far* or a combination of the two.

4.2.2. Ant Colony System

The Ant Colony System (ACS) (Dorigo & Gambardella, 1997; Gambardella & Dorigo, 1996) differs mainly by its pheromone update function. More in line with the natural behaviour of ants, ACS employs a *local pheromone update*, in addition to the pheromone update at the end of each epoch (*offline pheromone update*). Each ant performs the pheromone update after each construction step, applying pheromone to the last path traversed according to the following update function:

$$\tau_{ij} = (1-\varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0, \tag{7}$$

where $\varphi \in (0,1]$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

Similar to the MMAS algorithm, the offline pheromone update is applied at the end of each epoch by the *iteration-best* or the *best-so-far* ant only. The update function for the offline pheromone update is as follows:

$$\tau_{ij} = \begin{cases} (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij} & \text{if ant } (i,j) \text{ belongs to best tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

As in MMAS, $\Delta\tau_{ij} = 1/L_{best}$, where L_{best} can either be the *iteration-best* or the *best-so-far*.

In addition to the differences with the pheromone update procedure, the ACS also uses a different transition rule, called the *pseudorandom proportional* rule. If we let k be an ant located at state i, $q_0 \in [0,1]$ be a parameter, and q a random value in $[0,1]$, then the next node, j, is chosen according to the following probability distribution:

If $q \leqslant q_0$:

$$p_{ij}^k = \begin{cases} 1 & \text{if } j = argmax_{j \in N_k(i)} \tau_{ij} \cdot \eta_{ij}^{\beta} \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

else ($q > q_0$) and Eq. (3) is used.

4.2.3. Rank-based Ant System

The rank-based Ant System (AS_{rank}) (Bullnheimer, Hartl, & Strauss, 1996) incorporates the concept of ranking into the pheromone update procedure. The N ants are ranked according to the decreasing quality of their solutions, i.e. tour length ($L_1 \leqslant L_2 \leqslant \cdots \leqslant L_N$). The pheromone trials are updated offline in the form of a daemon action such that only the paths traversed by the $\sigma - 1$ best ants receive any pheromone, and the amount deposited depends directly on the ant's rank, μ, and on the quality of its solution. In addition, the paths traversed by the global-best solution, L^{gb}, receive an additional amount of pheromone which depends on the quality of that solution, weighted by parameter σ.

The pheromone update function is as follows:

$$\tau_{ij} = \rho\tau_{ij} + \Delta\tau_{ij}^{gb} + \Delta\tau_{ij}^{rank}, \tag{10}$$

where

$$\Delta\tau_{ij}^{rank} = \begin{cases} \sum_{\mu=1}^{\sigma-1} (\sigma - \mu) \frac{Q}{L_\mu} & \text{if the } \mu\text{th best ant travels} \\ & \text{on edge } (i,j) \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

and

$$\Delta\tau_{ij}^{gb} = \begin{cases} \sigma \frac{Q}{L^{gb}} & \text{if edge } (i,j) \text{ is part of the best solution found} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

5. Ant algorithms comparison

The results shown in Table 1 (taken from Stutzle et al., 2000) compare the performance of the four ant algorithms described above. This analysis is based on the performance of the individual algorithms as applied to a number of TSP instances. From these results we can see that for each of the three TSP instances considered here, the experiments carried out in Stutzle et al. (2000) show that the MMAS algorithm performs the best, followed by ACS, then AS_{rank}, and AS performing the worst out of the four.

The inclusion of these results here is merely meant to give an idea of change in performance such algorithmic modifications can yield. These results are of course subject to scrutiny, and the reader is directed to (Stutzle et al., 2000) for further details such as what parameter settings were used in these experiments.

6. Past and present applications

Ant algorithms have been successfully applied to many different problems in many different areas of scientific research and development, and are being used increasingly in industry, including successful implementation into 'real world' deployed systems.

Table 1
Computational results (taken from Stutzle et al. (2000)) showing the optimum results of four ant algorithms as applied to three different TSP instances. The first column gives the name of the TSP instance, the second column gives the true, known optimum solution, and the remaining four columns give the optimum results obtained with the respective ant algorithms.

Instance	Opt	MMAS	ACS	AS_{rank}	AS
eil51	426	427.6	428.1	434.5	437.3
kroA100	21,282	21320.3	21420.0	21746.0	22471.4
d198	15,780	15972.5	16054.0	16199.1	16702.1

This section details some of the more 'typical' applications to which ant algorithms and the ACO metaheuristic have been applied.

As previously stated, ant algorithms are particularly well suited to NP-hard combinatorial optimisation problems, and indeed the first problem to which the original AS algorithm was applied was the classic TSP problem, which itself is NP-hard. The TSP problem also has the characteristic of being a constrained shortest path problem, which is another important type of problem to which ant algorithms lend themselves particularly well to.

After the initial test-bed that was the TSP problem, ant algorithms were applied to other NP-hard problems such as the Quadratic Assignment problem (QAP) (Dorigo et al., 1996; Maniezzo, Colorni, & Dorigo, 1994) and the Job-Shop Scheduling (JSP) problem (Colorni, Dorigo, Maniezzo, & Trubian, 1994; Dorigo et al., 1996). The original AS was applied to the QAP and JSP to show its robustness, and shortly thereafter, improved ant algorithms were developed to solve these problems specifically. One of the next major applications for which ant algorithms were developed was the dynamic problem of data network routing (Schoonderwoerd, Holland, Bruten, & Rothkrantz, 1996; Di Caro & Dorigo, 1998), a shortest path problem where properties of the system such as node availability vary over time.

Other main applications of ant algorithms include the vehicle routing problem (Bullnheimer, Hartl, & Strauss, 1999; Gambardella & Taillard, 1999), graph colouring (Costa & Hertz, 1997) and set covering (Lessing, Dumitrescu, & Stutzle, 2004). The Vehicle routing problem in particular has strong links with industry; companies have developed tools built on ant algorithms that have been successfully deployed in real world working applications for various vehicle routing scenarios. Two specific example applications here include DyvOil and AntRoute (Gambardella et al., 2003; Rizzoli et al., 2003).

DyvOil is an application which supports planning the sales and distribution process of fuel oil. An *offline* module of DyvOil, which solves static vehicle routing problems by using ACO algorithms, is used every evening to plan vehicle tours for the next day. This offline planning module has been tested in a real world setting with a leading Swiss fuel oil distribution company called Pina Petroli. An increase of 20% upto 30% of vehicle routing performance has been observed using this technique over human generated plans (Rizzoli et al., 2003). Efficiency has also been shown when implementing ACS algorithms for dynamic vehicle routing problems in a similar scenario.

AntRoute is a similar application which has been designed for use in the supply chain of the Switzerland based supermarket known as Migros. An algorithm based on MACS-VRPTW (Gambardella & Taillard, 1999) is used to compute the tours of the distribution vehicles to supermarkets across Switzerland, and has again shown to outperform human planners.

The scope of applications in more recent years has increased significantly, with many more novel implementations of ant algorithms being developed. Aside from the highly specialised algorithms there are a number of broader areas of study to which ant algorithms are being applied. Examples of broad application areas typically seen in more recent trends include continuous optimisation (Socha, 2004; Socha & Blum, 2007) and parallel processing implementations (Manfrin, Birattari, Stutzle, & Dorigo, 2006; Talbi, Roux, Fonlupt, & Robillard, 1999). Continuous optimisation is a particularly interesting development for ant algorithms, since they were originally developed for discrete optimisation problems (Dorigo et al., 1999), and indeed it is this class of problem for which the ACO metaheuristic was developed.

7. Further developed ant algorithms

In more recent years, research and development involving ant algorithms has given rise to a vast array of specialised and novel ant algorithms, encompassing a much wider range of applications. The following examples aim to give a brief overview of a selection of more specialised ant algorithms, and an insight into the diversity in which ant algorithms may be used.

7.1. Ant algorithms for digital image processing

Digital image processing is well established amongst the scientific community and their exists many methods for performing various image processing tasks. Although certain aspects of machine learning and AI have been utilised in image processing for some time, the use of ant algorithms to perform image processing tasks is a relatively new technique. Ant algorithms have been used for basic low level image segmentation via boundary detection methods (Fernandes, Ramos, & Rosa, 2005; Nezamabadi-pour, Saryazdi, & Rashedi, 2006; Ramos & Almeida, 2000) and via clustering methods (Channa, Rajpoot, & Rajpoot, 2006; Ouadfel & Batouche, 2002).

The boundary detection algorithms make use of the pheromone aspect of ant algorithms in a novel way. The ants environment/search space is the digital image, such that the ants occupy pixels within the image, and move around the image in a discretised, pixel-wise fashion. The aim of the ants is to locate and map out the boundary's within the image. This is achieved by introducing heuristic information that weighs higher the probability of an ant moving from its current location to the allowed surrounding pixel that has the greatest boundary characteristics (greatest change in image gradient for example). Each ant deposits an amount of pheromone with each move to a new pixel, where the amount deposited may also be a function of, e.g. change in image gradient, and pheromone evaporation occurs at a fixed rate per iteration. The transition rule is then a function of the heuristic information and

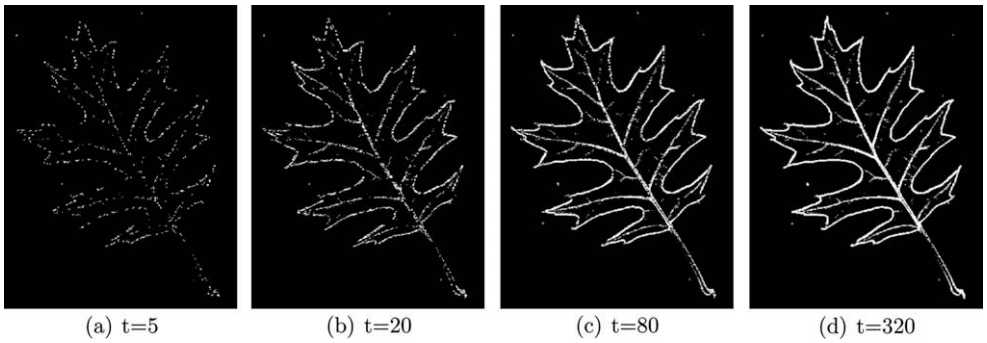

Fig. 2. Emerging pheromone map at different time-steps in the algorithm run. Brighter pixels equal higher pheromone concentration at that point.

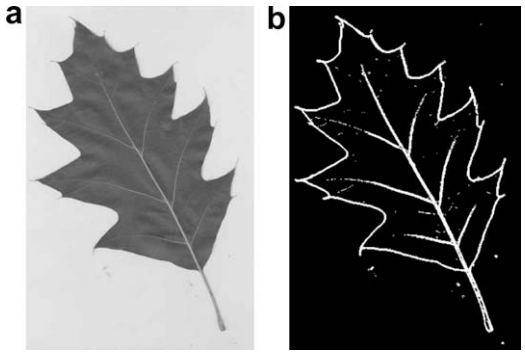

Fig. 3. Original leaf image (a) and final pheromone field map (b).

the pheromone map. A number of ants are started at random positions within the image and convergence toward the boundary areas within the image occurs, with the resulting pheromone trails mapping out the boundaries between image segments. Fig. 2 gives an example of ant agents converging on the edges/boundaries within a digital image of a leaf (Mullen, Monekosso, Barman, Remagnino, & Wilkin, 2008), showing the pheromone field at increasing time-steps, and Fig. 3 shows an example final pheromone field map of the leaf edges/boundaries after the algorithm has run, next to the original leaf image.

The premise of this algorithm has two major differences from the 'traditional' types of ant algorithms: individual ants never construct complete solutions of their own, and the pheromone is not only used to guide the ants movements but also represents the final solution in its entirety.

Unlike the boundary methods the clustering methods do not involve using the pheromone as a visual solution, but instead employ a more standard ant algorithm approach as a tool to optimise the mapping of pixels to clusters within the image. Other image processing ant algorithms include image thresholding (Malisia & Tizhoosh, 2006), another low level segmentation technique, where this time ants search the image for low grayscale regions, and in an area away from segmentation, ant algorithms have been used for image compression (Martinez, 2006).

7.2. Ant algorithms and the protein folding problem

Predicting the structure of protein formed from its linear sequence is an important problem in computational biology, yet it is NP-hard and thus very computationally expensive. Due to the high complexity of the problem, various models have been developed that simplify the search space of possible conformation. In particular the Hydrophobic–Polar (HP) model (Dill & Lau, 1989) involves the primary amino-acid sequences of a protein (which can be represented as a string over a 24 letter alphabet) being abstracted to a sequence of hydrophobic (H) and polar (P) residues that is represented as a string over the letters H and P. The protein conformations of this sequence are restricted to self-avoiding paths on a lattice. Ant algorithms have been proposed for both the 2D (Shmygelska & Hoos, 2003) and 3D (Findova, 2006) HP folding problem, which consider a 2D and 3D lattice respectively. The approach of (Shmygelska & Hoos, 2003) is based on the thermodynamic hypothesis which states that the native state of protein is the one with the lowest Gibbs free energy, where the energy of a conformation is calculated from the number of topological contacts between hydrophobic amino-acids that are not neighbours in a given sequence, such that for a conformation c with exactly n such H–H contacts, it has free energy $E(c) = n \cdot (-1)$.

In the 2D case candidate conformations are represented by the relative folding directions: *straight* (S), *left* (L), and *right* (R), which indicate the position of a given amino-acid relative to its immediate predecessor in the current sequence. In the construction phase of the algorithm each ant is placed at a random start point within the given protein sequence. Sequences are then built up by adding one amino-acid symbol at a time, with the relative direction in which the conformation is extended being chosen probabilistically in familiar ant algorithm fashion involving both heuristic and pheromone values. The heuristic values are tailored to guild the construction process towards conformations that maximise the number of H–H interactions, thus minimising the Gibbs free energy. The pheromone update equation is of the 'standard' form (Eq. (1)), where the relative solution quality is measured by: $E(c)/E^*$, where E^* is the known or approximated minimal energy for the given protein sequence. In this way solutions are built up in a way similar to how they are with the classic TSP application, where in this case the solution components are the folding directions, as opposed to paths between cites, as with TSP.

7.3. Ant algorithms and data mining

The overall goal of data mining is to extract knowledge from data (Fayyad, Piatetsky-Shapiro, & Smyth, 1996); discovering previously unknown, valid patterns and relationships in large data sets (Edelstein, 1999). Data mining algorithms have to perform such tasks as classification, clustering and forecasting, with the aim of analysing data as opposed to simply collecting it. Ant algorithms have been proposed that deal with the classification task of data mining (Holden & Freitas, 2004; Ji, Zhang, Liu, & Zhong, 2006; Parpinelli, Lopes, & Freitas, 2002; Smaldon & Freitas, 2006), where the goal is to assign each case (object, record, or instance) to one class, out of a set of predefined classes, based on the values of some attributes for the case.

More specifically, each classification rule takes on the form:

$$IF\ \langle term1\ AND\ term2\ AND\ \ldots\rangle\ THEN\ \langle class\rangle,$$

where each term is a triple $\langle attribute, operator, value\rangle$ such as $\langle gender = female\rangle$, with *value* being a value belonging to the domain of *attribute*, and the operator being a relational operator.

Ant-Miner follows a sequential covering approach to discover most, if not all, of the training cases (Parpinelli et al., 2002). Initially, the list of discovered rules is empty, and the training set is full. One classification rule is discovered for each iteration of the algorithm. This rule is then added to the list of discovered rules, and the training cases correctly covered by this rule are removed from the training set. The algorithm repeats while the number of uncovered training cases is greater than a user-specified threshold. For each iteration ants build solutions via three steps: rule construction, rule pruning and pheromone updating. An ant starts with an empty rule, and builds up partial rules by adding one term at a time, corresponding to the path (or segment of path) taken by the ant though the search space of terms. The choice of the next term to be added corresponds to the choice of direction to take to extend the current path. This choice depends on problem-dependent heuristic information as well as the amount of pheromone associated with each term. The ant keeps adding terms to the rule until either all attributes have been used, or the minimum number of cases covered per rule has been met. Once the rule has been constructed by the ant, it is then pruned of any irrelevant terms (see (Parpinelli et al., 2002) for more details) and the pheromone trails are updated.

The next ant then begins constructing its rule, using the updated pheromone trails as guidance. This process is repeated until all ants have constructed rules, or convergence has happened. Once

this process is halted an iteration is complete, and the best rule constructed by all ants is added to the list of discovered rules. Pheromone trails are then reinitialised to a constant amount and the process is repeated.

The heuristic information used in Ant-Miner involves a measure of entropy (amount of information) associated with each term. The pheromone update rule involves increasing the amount of pheromone associated with each term occurring, after pruning, in a rule found by an ant, where the increase is proportional to the quality of that rule. Additionally, pheromone evaporation is applied to each term that does not occur in the rule.

8. Ant algorithms and other machine learning techniques

Ant algorithms hold characteristics similar to many well established machine learning techniques, and indeed they are used to tackle similar problems. In this section we compare ant algorithms to a number of well established machine learning techniques, and also look at how ant algorithms can be combined with other machine learning techniques to produce novel, hybrid, algorithms with promising results.

8.1. Ant algorithms as an alternative to other machine learning techniques

Certain forms of ant algorithms, and in particularly ACO algorithms, show similarities with other machine learning techniques such as evolutionary computing and neural networks (Cordon et al., 2002; Dorigo et al., 1999). Below is a brief summary of some of the observed similarities and differences, highlighting where ant algorithms may be used in place of other well established machine learning techniques.

Both evolutionary computing and the ACO metaheuristic employ a population of individuals to incrementally build more suitable solutions to a given problem, by building on the solutions of previous populations. The main difference between the two is that with evolutionary computing, the knowledge of the problem is contained only within the current population, whereas the ACO metaheuristic continually uses information from a number of previous generation populations as its knowledge base, in the form of pheromone trials. There does exist however, specific forms of evolutionary computing algorithms that are more in line with the characteristics of the ACO metaheuristic. Further comparison with these specific algorithms can be found in Cordon et al. (2002) & Dorigo et al. (1999).

Artificial neural networks (Mitchell, 1997), like ant algorithms, are biologically inspired learning systems. Problems solved by both ant algorithms and neural networks can often be represented graphically by a set of connecting nodes. What we consider as 'states' in an ant algorithm is analogous with 'neurons' in a neural network. The local neighborhood structure around a given state is then equivalent to the set of synaptic-like links exiting the specific neuron (Dorigo et al., 1999). The ants are then associated with the input signals that propagate through the neural network. As connecting paths in ant algorithms receive more pheromone the more they are used, the more a synapse is used, the more its strength is increased. Also as with ant algorithms, synapses used to create better solutions are reinforced more than others.

The similarities seen in such algorithms makes it perhaps unsurprising that they are often used to solve similar types of problems. The differences however, mean that the performances of these algorithms differ from each other for different specific tasks. In many works such as the original AS (Dorigo et al., 1996) and ACS (Dorigo & Gambardella, 1997), ant algorithms have been tested on benchmark problems such as the TSP, and have yielded better solutions (i.e. closer to the true optimum) over other heuristic approaches. Also, in tests for rate of convergence towards the optimum solution, in many cases ant algorithms have proved to converge faster than other heuristic approaches. In other cases (Monekosso & Remagnino, 2001, 2004) it has been shown that by introducing certain elements particular to ant algorithms, such as the stigmergic effect via synthetic pheromones, to other machine learning techniques such a *quality learning* (Q-learning), this has increased performance over the original algorithm design (see Section 8.2.1).

8.2. Ant algorithms as a counterpart to other machine learning techniques

Two major machine learning techniques that have been used in conjunction with ant algorithms are Q-learning (Watkins, 2001) and *genetic algorithms* (Holland, 1992; Pilat & White, 2002; White, Pagurek, & Oppacher, 1998).

8.2.1. Ant algorithms and Q-learning

Q-learning falls within the category of reinforcement learning, which is a subset of machine learning to which one could also relate the concept of ant algorithms to. Reinforcement learning involves agents learning by trial and error which actions are best to take in their current environment in order to achieve their goals (Mitchell, 1997). In a training phase, each time an agent performs an action in its environment, it may receive a reward or penalty reflecting the desirability of the outcome of the action performed. The goal of the agent is then to choose sequences of actions that maximise the cumulative reward. More specifically, Q-learning involves learning an action-value function (where the action values are known as Q-values), which measures the utility (or 'goodness') of taking a given action in a given state within the environment. At each time-step, t, an agent in state s_i takes an action a which takes it to a new state s_{i+1}. The agent then receives a reward[3] r depending on the new state. The Q-values for each state-action pair are updated at each time-step until convergence between successive Q-values approaches zero, using the following equation:

$$Q_n(s_t,a) \leftarrow (1-\alpha_n)Q_{n-1}(s_t,a) + \alpha_n[r_t + \gamma \max_{a'} Q_{n-1}(s_{t+1},a')] \quad (13)$$

and

$$\alpha_n = \frac{1}{1+\text{visits}_n(s_t,a)}, \quad (14)$$

where γ is the discount factor, a' is the action that maximises Q, and visits(s_t,a) is the total number of times the given state-action pair have previously been visited.

An algorithm inspired by the original AS, called *Ant-Q*, was developed by Dorigo & Gambardella (1996), Gambardella & Dorigo (1995)). This algorithm has many similarities with the Q-learning algorithm, but also a few key differences; mainly that Ant-Q, unlike typical Q-learning algorithms, involves using multiple agents. These agents communicate, exchanging information in the form of AQ-values (which is the analogue of Q-values in Q-learning). As with AS, the Ant-Q algorithm was developed originally for the classic benchmark problem TSP. For TSP, $AQ(r,s)$ is the Ant-Q value associated with the path (r,s) between cities. $HE(r,s)$ is a heuristic value associated to path (r,s), which for TSP is the inverse of distance. k is an agent whose task it is to complete a closed tour of all cities, and associated with each agent k there is a list, $J_k(r)$, of all cities still to be visited, where r is the current city. This list acts as a kind of memory, and is another important difference between Ant-Q and Q-learning. The state transition rule for an agent k in city r is as follows:

[3] The reward is usually discounted into the future.

$$s = \begin{cases} argmax_{u \in J_k(r)} \left\{ [AQ(r,u)]^{\delta} \cdot [HE(r,u)]^{\beta} \right\} & \text{if } q \leqslant q_0, \\ S & \text{otherwise,} \end{cases} \quad (15)$$

where α and β are parameters which weigh the relative importance of the learned AQ-values and the heuristic values, q is a uniform probability randomly chosen value in [0,1], q_0 ($0 \leqslant q_0 \leqslant 1$) is a parameter such that the higher q_0 the smaller the probability to make a random choice, and S is a random variable selected according to a probability distribution given by the function of the $AQ(r,u)$'s and $HE(r,u)$'s, with $u \in J_r(r)$.

The update rule for the AQ-values is as follows:

$$AQ(r,s) \leftarrow (1-\alpha) \cdot AQ(r,s) + \alpha\left(\Delta AQ(r,s) + \gamma \cdot \mathrm{Max}_{z \in J_k(s)} AQ(s,z)\right), \quad (16)$$

where α and γ are the learning step and discount factor, respectively. This update rule is the same as in Q-learning except that the set of available actions in state s, i.e. the set $J_k(s)$, is a function of the previous history of agent k.

This approach adapts the idea of ant algorithms to that of Q-learning, and it is important to note that Ant-Q does not make use of artificial pheromones. A different approach has been developed in (Monekosso & Remagnino, 2001, 2004), which adapts the idea of Q-learning to that of ant algorithms, by introducing the use of artificial pheromones into multi-agent Q-learning.

The pheromone Q-learning (Phe-Q) algorithm uses the same Q-value update function as in Eq. (13), but with an additional factor to be maximised, called the belief factor. The belief factor is a function of the synthetic pheromone concentration on the trial and reflects the extent to which an agent will take into account the information laid down by other agents from the same cooperating group. The belief factor is the ratio between the sum of actual pheromone concentrations in the current plus surrounding states, and the sum of the maximum possible pheromone concentration in the current plus surrounding states, and is given by

$$B(s,a) = \frac{\sum_{s \in N_a} \Phi(s)}{\sum_{s \in N_a} \Phi_{\max}(\sigma)}, \quad (17)$$

where $\Phi(s)$ is the pheromone concentration at state s in the environment, and N_a is the set of surrounding states for a chosen action a.

With the addition of the belief factor the Q-learning update function then becomes

$$Q_n(s_t,a) \leftarrow (1-\alpha_n) Q_{n-1}(s_t,a) + \alpha_n \{ r_t + \gamma \max_{a'} [Q_{n-1}(s_{t+1},a') + \xi B(s_{t+1},a')] \}, \quad (18)$$

where the parameter ξ is a sigmoid function of time epochs $\geqslant 0$, such that it increases with the number of agents who successfully complete the given task.

In this example, where a key feature of the ant algorithms has been coupled with another established machine learning technique, improvements in the performance compared to the same algorithm without the additional ant algorithm feature have been shown (Monekosso & Remagnino, 2004). This is a clear example of how 'hybrid' algorithms, bringing elements of different machine learning techniques together, can produce superior performing algorithms.

8.2.2. Ant algorithms with genetic algorithms

Genetic algorithms are a branch of the wider field of study known as evolutionary computing, which is another subset of machine learning. The study of evolutionary computing, which began in the 1950s and 1960s, is inspired by biological evolution, where systems are developed that evolve a population of candidate solutions to a given problem, using operators based on natural selection and natural genetic variation (Mitchell, 1996).

The basic genetic algorithm involves iteratively updating a population of hypothesis. At each iteration, all individual hypothesis are evaluated against a problem specific fitness function. A new population is then created, first by probabilistically selecting a proportion of the most fit individuals. The remaining proportion of the new population is then created by probabilistically selecting pairs of the most fit individuals, and creating new offspring hypothesis by applying genetic crossover operators. In addition to this, genetic mutation operators may also be applied to random members of the new population before carrying it to the next iteration.

A genetic algorithm is used in Pilat and White (2002) and White et al. (1998) to automatically adapt the control parameters of ant algorithms. Choosing appropriate control parameters is vital to the success of the algorithm for the given application or problem, making sure an appropriate balance is met between exploration of unexplored areas of the search space/environment, and exploitation of previously learned preferences for states to visit, within the context of the problem. The use of genetic algorithms as described below could provide a useful automated alternative to the often used lengthly trial and error process of selecting appropriate control parameters, as well as potentially providing an adaptive online balance between exploration and exploitation within the search space/environment.

In Pilat and White (2002), the control parameters of the ant algorithm ACS are genetically adapted for the TSP as follows. Each ant is encoded with the parameters β,ρ and q_0 (see Section 4.2.2) in the form of bit strings (chromosomes). All chromosomes are initially randomised. At each iteration four ants are chosen via a tournament selection method. All four selected ants then construct their TSP tours. The algorithm then checks to see if a new tour has been found. The global update of the pheromone trail is carried out by the ant that produced the best overall tour, using that ants encoded ρ parameter. The fitness of each ant is calculated as the length of the tour found by that ant. The two best selected ants are crossed over to produce two offspring, which are then mutated. The worst two selected ants are then replaced by the two offspring. The idea here is to put pressure on the population to improve its performance, while keeping the population size constant.

Although there are clear benefits to be gained by using genetic algorithms with ant algorithms, their introduction can also bring about some problems. In Pilat and White (2002) it was shown that due to the high variability amongst the population of ants, many ants became trapped in local minima, with pheromone trials from good solutions being erased by worse performing ants, with no guarantee of finding the optimal solution. Conversely however, the introduction of the genetic algorithm was shown to increase the speed of convergence towards good solutions; a desirable characteristic for large problems where it is not desired, or indeed may not be possible to find the optimal solution.

As previously stated, choosing appropriate control parameters for a given ant algorithm is an important factor in its success with a given problem. In problems where the characteristics may change in some way over time, or problems that present themselves in different ways under different circumstances, it would be very advantageous to have some degree of automation for the fine tuning of the control parameters to suit the changing problem characteristics. The above presented method of using a genetic algorithm is one way of achieving this, and further research into similar methods could prove valuable in many applications of ant algorithms, especially those of a dynamic nature.

9. Concluding remarks

From the inspiration of social insects behaviour to solving complex computational problems, ant algorithms have progressed a long way. Ant algorithms are now being applied to a vast, wide ranging array of both theoretical and practical problems from many different areas of science and industry. From the original Ant System, through the development of the ACO metaheuristic, and beyond, there are now many different variations of ant algorithms, taking different adaptations on the original ideas based on the stigmergic behaviour of real ants in the wild. This paper has briefly detailed selected applications and developments of ant algorithms with the aim to show the diversity and also reinforce the success of this particular biologically inspired approach to computational problem solving.

The future will no doubt see further developments in application based ant algorithms. As described in this paper, the coupling of ant algorithms with other machine learning techniques can produce better performing 'hybrid' algorithms. It is envisaged that further development in this area will lead to learning algorithms that are more robust, and that will be especially useful in dynamic problems. Despite the success of ant algorithms however, formal theoretical explanations charting this success is limited (Dorigo, Birattari, & Stutzle, 2006), and this continues to be an area of research that holds for future work.

References

Bonabeau, B., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. New York, NY: Oxford University Press.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1996). A new rank-based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics, 7*(1), 25–38.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). An improved Ant System algorithm for the vehicle routing problem. *Annals of Operations Research, 89*, 319–328.

Channa, A. H., Rajpoot, N. M., & Rajpoot, K. M. (2006). Texture segmentation using ant tree clustering. In *2006 IEEE international conference on engineering of intelligent systems* (pp. 1–6).

Colorni, A., Dorigo, M., Maniezzo, V., & Trubian, M. (1994). Ant System for Job-Shop Scheduling. *JORBEL – Belgian Journal of Operations Research, Statistics and Computer Science, 34*(1), 39–53.

Cordon, O., Herrera, F., & Stutzle, T. (2002). A review on the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware and Soft Computing, 9*(2–3), :141–175.

Costa, D., & Hertz, A. (1997). Ants can colour graphs. *Journal of the Operational Research Society, 48*, 295–305.

Deneubourg, J. L., Aron, S., Goss, S., & Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior, 3*, 159.

Di Caro, G., & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research, 9*, 317–365.

Dill, K., & Lau, K. (1989). A lattice statistical mechanics model of the conformation sequence spaces of proteins. *Macromolecules, 22*, 3986–3997.

Dorigo, M., & Di Caro, G. (1999). The Ant Colony optimization metaheuristic. *New Ideas in Optimization* (pp. 11–32).

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*.

Dorigo, M., Di Caro, G., & Gambardella, M. (1999). Ant algorithms for discrete optimization. *Artificial Life, 5*(2), 137–172.

Dorigo, M., & Gambardella, L. M. (1996). A study of some properties of Ant-Q. In *PPSN IV: Proceedings of the 4th international conference on parallel problem solving from nature* (pp. 656–665). Springer-Verlag.

Dorigo, M., & Gambardella, L. M. (1997). Ant Colony System: A cooperating learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26*, 29–41.

Edelstein, H. A. (1999). *Introduction to data mining and knowledge discovery* (3rd ed.). Two Crows Corporation.

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery: An overview. In *Advances in knowledge discovery and data mining* (pp. 1–34). Cambridge, MA: AAAI/MIT.

Fernandes, C., Ramos, V., & Rosa, A. C. (2005). Self-regulated artificial ant colonies on digital image habitats. *International Journal of Lateral Computing, 2*(1), 1–8.

Findova, S. (2006). 3D protein folding problem using ant algorithm. In *Proceedings of BioPS international conference, Sofia, Bulgaria* (pp. 19–26).

Gambardella, L. M., Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the travelling salesman problem. In *Proceedings of ML-95, twelfth international conference on machine learning* (pp. 252–260). Tahoe City, CA: Morgan Kaufmann.

Gambardella, L. M., Taillard, E. D., & Agazzi, G. (1999). MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. *New Ideas in Optimization* (pp. 63–76).

Gambardella, L. M., & Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. *Proceedings of the 1996 IEEE international conference on evolutionary computation (ICEC'96)* (pp. 622–627). Piscataway, NJ: IEEE Press.

Gambardella, L. M., Rizzoli, A. E., Oliverio, F., Casagrande, N., Donati, A. V., Montemanni, R., et al. (2003). Ant colony optimization for vehicle routing in advanced logistic systems. In *MAS 2003 – international workshop on modelling and applied simulation, Bergeggi, Italy* (pp. 3–9).

Glover, F. (1989). Tabu search – Part I. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. (1990). Tabu search – Part II. *ORSA Journal on Computing, 2*(1), 4–32.

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften, 76*, 579–581.

Grasse, P. P. (1959). La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: Essai dinterpretation du comportement des termites constructeurs. *Insectes Sociaux, 6*, 41–81.

Holden, N., & Freitas, A. A. (2004). Web page classification with an ant colony algorithm. *Parallel Problem Solving from Nature – PPSN VIII, 3242*, 1092–1102.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press.

Ji, J., Zhang, N., Liu, C., & Zhong, N. (2006). An ant colony optimization algorithm for learning classification rules. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence* (pp. 1034–1037). Washington, DC, USA: IEEE Computer Society.

Kirkpatrick, S., Gelatt, C. D., Jr, & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Lessing, L., Dumitrescu, I., & Stutzle, T. (2004). A comparison between ACO algorithms for the set covering problem. *ANTS'2004, fourth international workshop on ant algorithms and swarm intelligence* (pp. 1–12). Springer-Verlag.

Lourenco, H. R., Martin, O., & Stuzle, T. (2002). Iterated local search. *Handbook of Metahearistics, 57*, 321–353.

Malisia, A. R., & Tizhoosh, H. R. (2006). Image thresholding using ant colony optimization. In *CRV '06: Proceedings of the 3rd Canadian conference on computer and robot vision (CRV'06)* (pp. 26). IEEE Computer Society.

Manfrin, M., Birattari, M., Stutzle, T., & Dorigo, M. (2006). Parallel ant colony optimization for the travelling salesman problem. In *Proceedings of ANTS 2006* (pp. 224–234). Springer-Verlag.

Maniezzo, V., Colorni, A., & Dorigo, M. (1994). The Ant System applied to the quadratic assignment problem. Technical Report IRIDIA/94-28.

Martinez, C. (2006). An ACO algorithmm for image compression. *Latinamerican Center for Informatics Studies Electronic Journal, 9*(2), 1–17.

Mitchell, M. (1996). *An introduction to genetic algorithms*. MIT Press.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Monekosso, N. D., & Remagnino, P. (2001). Phe-Q: A pheromone based Q-learning. In *AI 2001: Advances in artificial intelligence: 14th international joint conference on artificial intelligence, Adelaide, Australia* (pp. 1611–3349). Berlin: Springer.

Monekosso, N. D., & Remagnino, P. (2004). The analysis and performance evaluation of the pheromone-Q-learning algorithm. *Expert Systems, 21*(2), 80–91.

Mullen, R.J., Monekosso, D., Barman, S., Remagnino, P., & Wilkin, P. (2008). Artificial ants to extract leaf outlines and primary venation patterns. In *Proceedings of ANTS'2008 – 6th international workshop on ant algorithms* (pp. 251–258). Brussels, Belgium: Springer-Verlag.

Nezamabadi-pour, H., Saryazdi, S., & Rashedi, E. (2006). Edge detection using ant algorithms. *Soft Computing, 10*, 623–628.

Ouadfel, S., & Batouche, M. (2002). Unsupervised image segmentation using a colony of cooperating ants. In *BMCV '02: Proceedings of the second international workshop on biologically motivated computer vision* (pp. 109–116). Springer-Verlag.

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization – algorithms and complexity*. New York: Dover.

Parpinelli, R. S., Lopes, H. S., & Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation, 6*, 321–332.

Pilat, M. L., & White, T. (2002). Using genetic algorithms to optimize ACS-TSP. In *ANTS '02: Proceedings of the third international workshop on ant algorithms* (pp. 282–287). Springer-Verlag.

Ramos, V. & Almeida, F. (2000). Artificial ant colonies in digital image habitats – a mass behaviour effect study on pattern recognition. In *Proceedings of ANTS'2000 – 2nd international workshop on ant algorithms (from ant colonies to artificial ants)* (pp. 113–116).

Rizzoli, A. E., Casagrande, N., Donati, A. V., Gambardella, L. M., Lepori, D., Montemanni, et al. (2003). Planning and optimisation of vehicle routes for fuel oil distribution. In *MODSIM 2003 – Integrative modelling of biophysical, social and economic systems for resource management solutions, Townsville, Australia* (pp. 2024–2029).

Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunication networks. *Adaptive Behavior, 5*(2), 169–207.

Shmygelska, A., Hoos, H. H. (2003). An improved ant colony optimisation algorithm for the 2D HP protein folding problem. In *Advances in artificial intelligence: 16th Conference of the Canadian society for computational studies of intelligence, Halifax, Canada, AI 2003* (page 993).

Smaldon, J., & Freitas, A. A. (2006). A new version of the Ant-Miner algorithm discovering unordered rule sets. In *GECCO '06: Proceedings of the 8th annual conference on genetic and evolutionary computation* (pp. 43–50). New York, NY, USA: ACM.

Socha, K. (2004). ACO for continuous and mixed-variable optimization. In *Ant colony optimization and swarm intelligence, 4th international workshop, ANTS 2004* (pp. 25–36). Springer-Verlag.

Socha, K., & Blum, C. (2007). An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing and Applications, 16*(3), 235–247.

Stutzle, T., & Hoos, H. H. (2000). Max–min ant system. *Future Generation Computer Systems, 16*(8), 889–914.

Talbi, E. G., Roux, O., Fonlupt, C., & Robillard, D. (1999). Parallel ant colonies for combinatorial optimization problems. In *Parallel and distributed processing, 11 IPPS/SPDP'99 workshops* (vol. 1586, pp. 239–247).

Watkins, C. J. C. H. (2001). Learning with delayed rewards. PhD thesis, Psychology Department, University of Cambridge, England.

White, T., Pagurek, B., & Oppacher, F. (1998). ASGA: Improving the ant system by integration with genetic algorithms. In *Programming 1998: Proceedings of the third annual conference* (pp. 610–617). Madison, Wisconsin, USA: University of Wisconsin, Morgan Kaufmann.