Technical report 10-029

Ant colony learning algorithm for optimal control[*]

J.M. van Ast, R. Babuška, and B. De Schutter

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.51.19 (secretary)
fax: +31-15-278.66.79
URL: http://www.dcsc.tudelft.nl

[*] This report can also be downloaded via http://pub.deschutter.info/abs/10_029.html

Ant Colony Learning Algorithm for Optimal Control

Jelmer Marinus van Ast, Robert Babuška, and Bart De Schutter

Abstract Ant Colony Optimization (ACO) is an optimization heuristic for solving combinatorial optimization problems and it is inspired by the swarming behavior of foraging ants. ACO has been successfully applied in various domains, such as routing and scheduling. In particular, the agents, called ants here, are very efficient at sampling the problem space and quickly finding good solutions. Motivated by the advantages of ACO in combinatorial optimization, we develop a novel framework for finding optimal control policies that we call Ant Colony Learning (ACL). In ACL, the ants all work together to collectively learn optimal control policies for any given control problem for a system with nonlinear dynamics. In this chapter, we will discuss the ACL framework and its implementation with crisp and fuzzy partitioning of the state space. We demonstrate the use of both versions in the control problem of two-dimensional navigation in an environment with variable damping and discuss their performance.

1 Introduction

Ant Colony Optimization (ACO) is a metaheuristic for solving combinatorial optimization problems [1]. Inspired by ants and their behavior in finding shortest paths from their nest to sources of food, the virtual ants in ACO aim at jointly finding opti-

Jelmer Marinus van Ast
Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands, e-mail: j.m.vanast@tudelft.nl

Robert Babuška
Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands e-mail: r.babuska@tudelft.nl

Bart De Schutter
Delft Center for Systems and Control & Marine and Transport Technology, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands e-mail: b@deschutter.info

mal paths in a given search space. The key ingredients in ACO are the pheromones. With real ants, these are chemicals deposited by the ants and their concentration encodes a map of trajectories, where stronger concentrations represent the trajectories that are more likely to be optimal. In ACO, the ants read and write values to a common pheromone matrix. Each ant autonomously decides on its actions biased by these pheromone values. This indirect form of communication is called stigmergy. Over time, the pheromone matrix converges to encode the optimal solution of the combinatorial optimization problem, but the ants typically do not all converge to this solution, thereby allowing for constant exploration and the ability to adapt the pheromone matrix to changes in the problem structure. These characteristics have resulted in a strong increase of interest in ACO over the last decade since its introduction in the early nineties [2].

The Ant System (AS), which is the basic ACO algorithm, and its variants, have successfully been applied to various optimization problems, such as the traveling salesman problem [3], load balancing [4], job shop scheduling [5, 6], optimal path planning for mobile robots [7], and routing in telecommunication networks [8]. An implementation of the ACO concept of pheromone trails for real robotic systems is described in [9]. A survey of industrial applications of ACO is presented in [10]. An overview of ACO and other metaheuristics to stochastic combinatorial optimization problems can be found in [11].

This chapter presents the extension of ACO to the learning of optimal control policies for continuous-time, continuous-state dynamic systems. We call this new class of algorithms Ant Colony Learning (ACL) and present two variants, one with a crisp partitioning of the state space and one with a fuzzy partitioning. The work in this chapter is based on [12] and [13] in which we have introduced these respective variants of ACL. This chapter not only unifies the presentation of both versions, it also discusses them in greater detail, both theoretically and practically. Crisp ACL, as we call ACL with crisp partitioning of the state space, is the more straightforward extension of classical ACO to solving optimal control problems. The state space is quantized in a crisp manner, such that each value of the continuous-valued state is mapped to exactly one bin. This renders the control problem non-deterministic, as state transitions are measured in the quantized domain, but originate from the continuous domain. This makes the control problem much more difficult to solve. Therefore, the variant of ACL in which the state space is partitioned with fuzzy membership functions was developed. We call this variant Fuzzy ACL. With a fuzzy partitioning, the continuous-valued state is mapped to the set of membership functions. The state is a member of each of these membership functions to a certain degree. The resulting Fuzzy ACL algorithm is somewhat more complicated compared to Crisp ACL, but there are no non-deterministic state transitions introduced.

One of the first real applications of the ACO framework to optimization problems in continuous search spaces is described in [14] and [15]. An earlier application of the ant metaphor to continuous optimization appears in [16], with more recent work like the Aggregation Pheromones System in [17] and the Differential Ant-Stigmergy Algorithm in [18]. [19] is the first work linking ACO to optimal control. Although presenting a formal framework, called *ant programming*, no application, or study of its performance is presented. Our algorithm shares some similarities with the Q-learning algorithm. Earlier work [20] introduced the Ant-Q algorithm, which is the most notable other work relating ACO with Q-learning. However, Ant-Q has been developed for combinatorial optimization problems and not to optimal control problems. Because of this difference, ACL is novel in all major structural aspects of the Ant-Q algorithm, namely the choice of the action selection method, the absence of the heuristic variable, and the choice of the set of ants used in the update. There are only a few publications that combine ACO with the concept of fuzzy control [21, 22, 23]. In all three publications fuzzy controllers are obtained using ACO, rather than developing an actual fuzzy ACO algorithm, as introduced in this paper.

This chapter is structured as follows. In Section 2, the ACO heuristic is reviewed with special attention paid to the Ant System and the Ant Colony System, which are among the most popular ACO algorithms. Section 3 presents the general layout of ACL, with in more detail its crisp and fuzzy version. Section 4 presents the application of both ACL algorithms on the control problem of two-dimensional vehicle navigation in an environment with a variable damping profile. The learning performance of Crisp and Fuzzy ACL is studied using of a set of performance measures and the resulting policies are compared to an optimal policy obtained by the fuzzy Q-iteration algorithm from [24]. Section 5 concludes this chapter and presents and outline for future research.

2 Ant Colony Optimization

This section presents the preliminaries for understanding the ACL algorithm. It presents the framework of all ACO algorithms in Section 2.1. The Ant System, which is the most basic ACO algorithm, is discussed in Section 2.2 and the Ant Colony System, which is slightly more advanced compared to the Ant System, is discussed in Section 2.3. ACL is based on these two algorithms.

2.1 ACO Framework

ACO algorithms have been developed to solve hard combinatorial optimization problems [1]. A combinatorial optimization problem can be represented as a tuple $P = \langle \mathcal{S}, F \rangle$, where $\mathcal{S}$ is the solution space with $s \in \mathcal{S}$ a specific candidate solution and where $F : \mathcal{S} \to \mathbb{R}_+$ is a fitness function assigning strictly positive values to candidate solutions, where higher values correspond to better solutions. The purpose of the algorithm is to find a solution $s^* \in \mathcal{S}$, or a set of solutions $\mathcal{S}^* \subseteq \mathcal{S}$ that maximize the fitness function. The solution s^* is then called an optimal solution and $\mathcal{S}^*$ is called the set of optimal solutions.

In ACO, the combinatorial optimization problem is represented by a graph consisting of a set of vertices and a set of arcs connecting the vertices. A particular solution s is a concatenation of solution components (i,j), which are pairs of vertices i and j connected by the arc ij. The concatenation of solution components forms a path from the initial vertex to the terminal vertex. This graph is called the *construction* graph, as the solutions are constructed incrementally by moving over the graph. How the terminal vertices are defined depends on the problem considered. For instance, in the traveling salesman problem[1], there are multiple terminal vertices, namely for each ant the terminal vertex is equal to its initial vertex, after visiting all other vertices exactly once. For the application to control problems, as considered in this chapter, the terminal vertex corresponds to the desired state of the system. Two values are associated with the arcs: a pheromone trail variable τ_{ij} and a heuristic variable η_{ij}. The pheromone trail represents the acquired knowledge about the optimal solutions over time and the heuristic variable provides a priori information about the quality of the solution component, i.e., the quality of moving from vertex i to vertex j. In the case of the traveling salesman problem, the heuristic variables typically represent the inverse of the distance between the respective pair of cities. In general, a heuristic variable represents a short-term quality measure of the solution component, while the task is to acquire a concatenation of solution components that overall form an optimal solution. A pheromone variable, on the other hand, encodes the measure of the long-term quality of concatenating the respective solution components.

2.2 The Ant System

The most basic ACO algorithm is called the Ant System (AS) [25] and works as follows. A set of M ants is randomly distributed over the vertices. The heuristic variables η_{ij} are set to encode the prior knowledge by favoring the choice of some vertices over others. For each ant c, the partial solution $s_{\mathrm{p},c}$ is initially empty and all pheromone variables are set to some initial value $\tau_0 > 0$. In each iteration, each ant decides based on some probability distribution, which solution component (i,j) to add to its partial solution $s_{\mathrm{p},c}$. The probability $p_c\{j|i\}$ for an ant c on a vertex i to move to a vertex j within its feasible neighborhood $\mathcal{N}_{i,c}$ is defined as:

$$p_c\{j|i\} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in\mathcal{N}_{i,c}}\tau_{il}^{\alpha}\eta_{il}^{\beta}}, \forall j \in \mathcal{N}_{i,c}, \tag{1}$$

with $\alpha \geq 1$ and $\beta \geq 1$ determining the relative importance of η_{ij} and τ_{ij} respectively. The feasible neighborhood $\mathcal{N}_{i,c}$ of an ant c on a vertex i is the set of not yet visited vertices that are connected to i. By moving from vertex i to vertex j, ant

[1] In the traveling salesman problem, there is a set of cities connected by roads of different lengths and the problem is to find the sequence of cities that takes the traveling salesman to all cities, visiting each city exactly once and bringing him back to its initial city with a minimum length of the tour.

c adds the associated solution component (i, j) to its partial solution $s_{\mathrm{p},c}$ until it reaches its terminal vertex and completes its candidate solution.

The candidate solutions of all ants are evaluated using the fitness function $F(s)$ and the resulting value is used to update the pheromone levels by:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{s \in \mathcal{S}_{\mathrm{upd}}} \varDelta\tau_{ij}(s), \tag{2}$$

with $\rho \in (0, 1)$ the evaporation rate and $\mathcal{S}_{\mathrm{upd}}$ the set of solutions that are eligible to be used for the pheromone update, which will be explained further on in this section. This update step is called the *global* pheromone update step. The pheromone deposit $\varDelta\tau_{ij}(s)$ is computed as:

$$\varDelta\tau_{ij}(s) = \begin{cases} F(s) & \text{, if } (i,j) \in s \\ 0 & \text{, otherwise.} \end{cases}$$

The pheromone levels are a measure of how desirable it is to add the associated solution component to the partial solution. In order to incorporate forgetting, the pheromone levels decrease by some factor in each iteration. This is called pheromone evaporation in correspondence to the physical evaporation of the chemical pheromones for real ant colonies. By evaporation, it can be avoided that the algorithm prematurely converges to suboptimal solutions. Note that in (2) the pheromone level on all vertices is evaporated and only those vertices that are associated with the solutions in the update set receive a pheromone deposit.

In the following iteration, each ant repeats the previous steps, but now the pheromone levels have been updated and can be used to make better decisions about which vertex to move to. After some stopping criterion has been reached, the values of τ and η on the graph encode the solution for all (i, j)-pairs. This final solution can be extracted from the graph as follows:

$$(i, j): \quad j = \arg\max_{l}(\tau_{il}^{\alpha}\eta_{il}^{\beta}), \quad \forall i.$$

Note that all ants are still likely to follow suboptimal trajectories through the graph, thereby exploring constantly the solution space and keeping the ability to adapt the pheromone levels to changes in the problem structure.

There exist various rules to construct $\mathcal{S}_{\mathrm{upd}}$, of which the most standard one is to use all the candidate solutions found in the trial. This update set is then called: $\mathcal{S}_{\mathrm{trial}}$[2]. This update rule is typical for the Ant System. However, other update rules have shown to outperform the AS update rule in various combinatorial optimization

[2] In ACO literature, the term trial is seldom used. It is rather a term from the reinforcement learning (RL) community [26]. In our opinion it is also a more appropriate term for ACO, especially in the setting of optimal control, and we will use it to denote the part of the algorithm from the initialization of the ants over the state space until the global pheromone update step. The corresponding term for a trial in the ACO literature is iteration and the set of all candidate solutions found in each iteration is denoted as $\mathcal{S}_{\mathrm{iter}}$. In this chapter, equivalently to RL, we prefer to use the word iteration to indicate one step of feeding an input to the system and measuring its state. In Figure 2,

problems. Rather than using the complete set of candidate solutions from the last trial, either the best solution from the last trial, or the best solution since the initialization of the algorithm can be used. The former update rule is called *Iteration Best* in the literature (which should be called *Trial Best* in our terminology), and the latter is called *Best-So-far*, or *Global Best* in the literature [1]. These methods result in a strong bias of the pheromone trail reinforcement towards solutions that have been proven to perform well and additionally reduce the computational complexity of the algorithm. As the risk exists that the algorithm prematurely converges to suboptimal solutions, these methods are only superior to AS if extra measures are taken to prevent this, such as the local pheromone update rule, explained in Section 2.3. Two of the most successful ACO variants in practice that implement the update rules mentioned above, are the Ant Colony System [27] and the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [28]. Because of its relation to ACL, we will explain the Ant Colony System next.

2.3 The Ant Colony System

The Ant Colony System (ACS) [27] is an extension to the AS and is one of the most successful and widely used ACO algorithms. There are four main differences between the AS and the ACS:

1. The ACS uses the global-best update rule in the global pheromone update step. This means that only the one solution that has been found since the start of the algorithm that has the highest fitness, called s_{gb} is used to update the pheromone variables at the end of the trial. This is a form of elitism in ACO algorithms that has shown to significantly speed up the convergence to the optimal solution.
2. The global pheromone update is only performed for the (i, j)-pairs that are an element of the global best solution. This means that not all pheromone levels are evaporated, as with the AS, but only those that also receive a pheromone deposit.
3. The pheromone deposit is weighted by ρ. As a result of this and the previous two differences, the global pheromone update rule is:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \rho\mathit{\Delta}\tau_{ij}(s_{\mathrm{gb}}) & \text{, if } (i,j) \in s_{\mathrm{gb}} \\ \tau_{ij} & \text{, otherwise.} \end{cases}$$

4. An important element from the ACS algorithm that acts as a measure to avoid premature convergence to suboptimal solutions is the local pheromone update step, which occurs for each ant after each iteration within a trial and is defined as follows:

$$\tau_{ij}(\kappa+1) = (1-\gamma)\tau_{ij}(\kappa) + \gamma\tau_0, \tag{3}$$

everything that happens in the outer loop is the trial and the inner loops represent the iterations for all ants in parallel.

where $\gamma \in (0,1)$ is a small parameter similar to ρ, ij is the index corresponding to the (i,j)-pair just added to the partial solution, and τ_0 is the initial value of the pheromone trail. The effect of (3) is that during the trial, the visited solution components are made less attractive for other ants to take, in that way promoting the exploration of other, less frequently visited, solution components.

5. There is an explicit exploration-exploitation step with the selection of the next node j, where with a probability of ϵ, j is chosen as being the node with the highest value of $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ (exploitation) and with the probability $(1-\epsilon)$, a random action is chosen according to (1) (exploration).

The Ant Colony Learning algorithm, which is presented next, is based on the AS combined with the local pheromone update step from the ACS algorithm.

3 Ant Colony Learning

This section presents the Ant Colony Learning (ACL) class of algorithms. First, in Section 3.1 the optimal control problem for which ACL has been developed is introduced. Section 3.2 presents the general layout of ACL. ACL with crisp state space partitioning is presented in Section 3.3 and the fuzzy variant in Section 3.4. Regarding the practical application of ACL, Section 3.5 discusses ways of setting the parameters from the algorithm and Section 3.6 presents a discussion on the relation of ACL to reinforcement learning.

3.1 The Optimal Control Problem

Assume that we have a nonlinear dynamic system, characterized by a continuous-valued state vector $\mathbf{x} = \begin{bmatrix} x_1 \; x_2 \; \ldots \; x_n \end{bmatrix}^{\mathrm{T}} \in \mathcal{X} \subseteq \mathbb{R}^n$, with $\mathcal{X}$ the state space and n the order of the system. Also assume that the state can be controlled by an input $\mathbf{u} \in \mathcal{U}$ that can only take a finite number of values and that the state can be measured at discrete time steps, with a sample time T_{s} with k the discrete time index. The sampled system is denoted as:

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)). \tag{4}$$

The optimal control problem is to control the state of the system from any given initial state $\mathbf{x}(0) = \mathbf{x}_0$ to a desired goal state $\mathbf{x}(K) = \mathbf{x}_{\mathrm{g}}$ in at most K steps and in an optimal way, where optimality is defined by minimizing a certain cost function. As an example, take the following quadratic cost function:

$$J(s) = J(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}) = \sum_{k=0}^{K-1} \mathbf{e}^{\mathrm{T}}(k+1)\mathbf{Q}\mathbf{e}(k+1) + \mathbf{u}^{\mathrm{T}}(k)\mathbf{R}\mathbf{u}(k), \tag{5}$$

with s the solution found by a given ant, $\tilde{\mathbf{x}} = \mathbf{x}(1) \ldots \mathbf{x}(K)$ and $\tilde{\mathbf{u}} = \mathbf{u}(0) \ldots \mathbf{u}(K-1)$ the sequences of states and actions in that solution respectively, $\mathbf{e}(k+1) = \mathbf{x}(k+1) - \mathbf{x}_\mathrm{g}$ the error at time $k+1$, and $\mathbf{Q}$ and $\mathbf{R}$ positive definite matrices of appropriate dimensions. The problem is to find a nonlinear mapping from the state to the input $\mathbf{u}(k) = \mathbf{g}(\mathbf{x}(k))$ that, when applied to the system in $\mathbf{x}_0$ results in a sequence of state-action pairs $(\mathbf{u}(0), \mathbf{x}(1))$, $(\mathbf{u}(1), \mathbf{x}(2))$, ..., $(\mathbf{u}(K-1), \mathbf{x}_\mathrm{g})$ that minimizes this cost function. The function $\mathbf{g}$ is called the control policy. We make the assumption that $\mathbf{Q}$ and $\mathbf{R}$ are selected in such a way that $\mathbf{x}_\mathrm{g}$ is reached in at most K time steps. The matrices $\mathbf{Q}$ and $\mathbf{R}$ balance the importance of speed versus the aggressiveness of the controller. This kind of cost function is frequently used in optimal control of linear systems, as the optimal controller minimizing the quadratic cost can be derived as a closed expression after solving the corresponding Riccati equation using the $\mathbf{Q}$ and $\mathbf{R}$ matrices and the matrices of the linear state space description [29]. In our case, we aim at finding control policies for non-linear systems which in general cannot be derived analytically from the system description and the $\mathbf{Q}$ and $\mathbf{R}$ matrices. Note that our method is not limited to the use of quadratic cost functions.

The control policy we aim to find with ACL will be a state feedback controller. This is a reactive policy, meaning that it will define a mapping from states to actions without the need of storing the states (and actions) of previous time steps. This poses the requirement on the system that it can be described by a state-transition mapping for a quantized state $\mathbf{q}$ and an action (or input) $\mathbf{u}$ in discrete time as follows:

$$\mathbf{q}(k+1) \sim p(\mathbf{q}(k), \mathbf{u}(k)), \tag{6}$$

with p a probability distribution function over the state-action space. In this case, the system is said to be a Markov Decision Process (MDP) and the probability distribution function p is said to be the Markov model of the system. Note that finding an optimal control policy for an MDP is equivalent to finding the optimal sequence of state-action pairs from any given initial state to a certain goal state, which is a combinatorial optimization problem. When the state transitions are stochastic, like in (6), it is a stochastic combinatorial optimization problem. As ACO algorithms are especially applicable to (stochastic) combinatorial optimization problems, the application of ACO to deriving control policies is evident.

3.2 General Layout of ACL

ACL can be categorized as a model-free learning algorithm, as the ants do not have direct access to the model of the system and must learn the control policy just by interacting with it. Note that as the ants interact with the system in parallel, the implementation of this algorithm to a real system requires multiple copies of this system, one for each ant. This hampers the practical applicability of the algorithm to real systems. However, when a model of the real system is available, the parallelism can take place in software. In that case, the learning happens off-line and after convergence, the resulting control policy can be applied to the real system. In principle, the ants could also interact with one physical system, but this would require either a serial implementation in which each ant performs a trial and the global pheromone update takes place after the last ant has completed its trial, or a serial implementation in which each ant only performs a single iteration, after which the state of the system must be reinitialized for the next ant. In the first case, the local pheromone update will lose most of its use, while in the second case, the repetitive reinitializations will cause a strong increase of simulation time and heavy load on the system. In either case, the usefulness of ACL will be heavily compromised.

An important consideration for an ACO approach to optimal control is which set of solutions to use in the global pheromone update step. When using the Global Best pheromone update rule in an optimal control problem, all ants have to be initialized to the same state, as starting from states that require less time and less effort to reach the goal would always result in a better Global Best solution. Ultimately, initializing an ant exactly in the goal state would be the best possible solution and no other solution, starting from more interesting states would get the opportunity to update the pheromones in the global pheromone update phase. In order to find a control policy from *any* initial state to the goal state, the Global Best update rule cannot be used. By simply using all solutions of all ants in the updating, like in the original AS algorithm, the resulting algorithm does allow for random initialization of the ants over the state space and is therefore used in ACL.

3.3 ACL with Crisp State Space Partitioning

For a system with a continuous-valued state space, optimization algorithms like ACO can only be applied if the state space is quantized. The most straightforward way to do this is to divide the state space into a finite number of bins, such that each state value is assigned to exactly one bin. These bins can be enumerated and used as the vertices in the ACO construction graph.

3.3.1 Crisp State Space Partitioning

Assume a continuous-time, continuous-state system sampled with a sample time T_s. In order to apply ACO, the state $\mathbf{x}$ must be quantized into a finite number of bins to get the quantized state $\mathbf{q}$. Depending on the sizes and the number of these bins, portions of the state space will be represented with the same quantized state. One can imagine that applying an input to the system that is in a particular quantized state results in the system to move to a next quantized state with some probability. In order to illustrate these aspects of the quantization, we consider the continuous model to be cast as a discrete stochastic *automaton*. An automaton is defined by the triple $\Sigma = (\mathcal{Q}, \mathcal{U}, \phi)$, with $\mathcal{Q}$ a finite or countable set of discrete states, $\mathcal{U}$ a finite or countable set of discrete inputs, and $\phi : \mathcal{Q} \times \mathcal{U} \times \mathcal{Q} \rightarrow [0,1]$ a state transition function.

Given a discrete state vector $\mathbf{q} \in \mathcal{Q}$, a discrete input symbol $\mathbf{u} \in \mathcal{U}$, and a discrete next state vector $\mathbf{q}' \in \mathcal{Q}$, the (Markovian) state transition function ϕ defines the probability of this state transition, $\phi(\mathbf{q}, \mathbf{u}, \mathbf{q}')$, making the automaton stochastic. The probabilities over all states $\mathbf{q}'$ must each sum up to one for each state-action pair $(\mathbf{q}, \mathbf{u})$. An example of a stochastic automaton is given in Figure 1. In this figure, it is clear that, e.g., applying an action $u = 1$ to the system in $q = 1$ can move the system to a next state that is either $q' = 1$ with a probability of 0.2, or $q' = 2$ with a probability of 0.8.

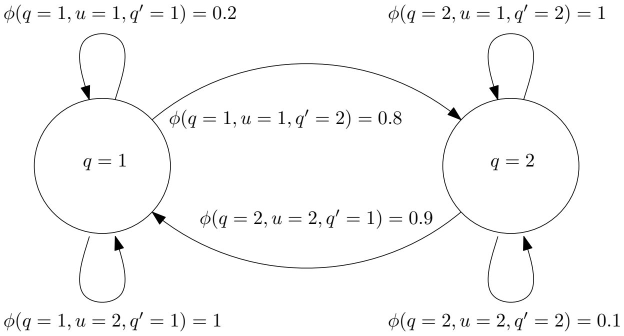

$\phi(q = 1, u = 1, q' = 1) = 0.2$
$\phi(q = 2, u = 1, q' = 2) = 1$
$\phi(q = 1, u = 1, q' = 2) = 0.8$
$q = 1$
$q = 2$
$\phi(q = 2, u = 2, q' = 1) = 0.9$
$\phi(q = 1, u = 2, q' = 1) = 1$
$\phi(q = 2, u = 2, q' = 2) = 0.1$

Fig. 1 An example of a stochastic automaton.

The probability distribution function determining the transition probabilities reflects the system dynamics and the set of possible control actions is reflected in the structure of the automaton. The probability distribution function can be estimated from simulations of the system over a fine grid of pairs of initial states and inputs, but for the application of ACL, this is not necessary. The algorithm can directly interact with the continuous-state dynamics of the system, as will be described in the following section.

3.3.2 Crisp ACL Algorithm

At the start of every trial, each ant is initialized with a random continuous-valued state of the system. This state is quantized onto $\mathcal{Q}$. Each ant has to choose an action and add this state-action pair to its partial solution. It is not known to the ant to which state this action will take him, as in general there is a set of next states to which the ant can move, according to the system dynamics.

No heuristic values are associated with the vertices, as we assume in this chapter that no a priori information is available about the quality of solution components. This is implemented by setting all heuristic values equal to one. It can be seen that

η_{ij} disappears from (1) in this case. Only the value of α remains as a tuning parameter, now in fact only determining the amount of exploration as higher values of α make the probability higher for choosing the action associated with the largest pheromone level. In the ACS, there is an explicit exploration-exploitation step when the next node j associated with the highest value of $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ is chosen with some probability ϵ (exploitation) and a random node is chosen with the probability $(1-\epsilon)$ according to (1) (exploration). In our algorithm, as we do not have the heuristic factor η_{ij}, the amount of exploration over exploitation can be tuned by the variable α. For this reason, we do not include an explicit exploration-exploitation step based on ϵ. Without this and without the heuristic factor, the algorithm needs less tuning and is easier to apply.

Action Selection

The probability of an ant c to choose an action $\mathbf{u}$ when in a state $\mathbf{q}_c$ is:

$$p_c\{\mathbf{u}|\mathbf{q}_c\} = \frac{\tau_{\mathbf{q}_c\mathbf{u}}^{\alpha}}{\sum_{\mathbf{l}\in\mathcal{U}_{\mathbf{q}_c}} \tau_{\mathbf{q}_c\mathbf{l}}^{\alpha}}, \tag{7}$$

where $\mathcal{U}_{\mathbf{q}_c}$ is the action set available to ant c in state $\mathbf{q}_c$.

Local Pheromone Update

The pheromones are initialized equally for all vertices and set to a small positive value τ_0. During every trial, all ants construct their solutions in parallel by interacting with the system until they either have reached the goal state, or the trial exceeds a certain pre-specified number of iterations $K_{\max}$. After every iteration, the ants perform a local pheromone update, equal to (3), but in the setting of Crisp ACL:

$$\tau_{\mathbf{q}_c\mathbf{u}_c}(k+1) \leftarrow (1-\gamma)\tau_{\mathbf{q}_c\mathbf{u}_c}(k) + \gamma\tau_0. \tag{8}$$

Global Pheromone Update

After completion of the trial, the pheromone levels are updated according to the following global pheromone update step:

$$\begin{aligned}\tau_{\mathbf{qu}}(\kappa+1) \leftarrow & (1-\rho)\tau_{\mathbf{qu}}(\kappa) && (9)\\ & + \rho \sum_{\substack{s\in\mathcal{S}_{\mathrm{trial}};\\(\mathbf{q},\mathbf{u})\in s}} J^{-1}(s), \quad \forall(\mathbf{q},\mathbf{u}) : \exists s \in \mathcal{S}_{\mathrm{trial}}; (\mathbf{q},\mathbf{u}) \in s, && (10)\end{aligned}$$

with $\mathcal{S}_{\mathrm{trial}}$ the set of all candidate solutions found in the trial and κ the trial counter. This type of update rule is comparable to the AS update rule, with the important difference that only the pheromone levels are evaporated that are associated with the elements in the update set of solutions. The pheromone deposit is equal to $J^{-1}(s) = J^{-1}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}})$, the inverse of the cost function over the sequence of quantized state-action pairs in s according to (5).

The complete algorithm is given in Algorithm 1. In this algorithm the assignment $\mathbf{x}_c \leftarrow \mathrm{random}(\mathcal{X})$ in Step 7 selects for ant c a random state $\mathbf{x}_c$ from the state space

$\mathcal{X}$ with a uniform probability distribution. The assignment $\mathbf{q}_c \leftarrow \text{quantize}(\mathbf{x}_c)$ in Step 9 quantizes for ant c the state $\mathbf{x}_c$ into a quantized state $\mathbf{q}_c$ using the prespecified quantization bins from $\mathcal{Q}$. A flow diagram of the algorithm is presented in Figure 2. In this figure, everything that happens in the outer loop is the trial and the inner loops represent the iterations for all ants in parallel.

Algorithm 1 The Ant Colony Learning algorithm for optimal control problems.

```
Input: $\mathcal{Q}, \mathcal{U}, \mathcal{X}, \mathbf{f}, M, \tau_0, \rho, \gamma, \alpha, K_{\max}, \kappa_{\max}$
 1: $\kappa \leftarrow 0$
 2: $\tau_{\mathbf{qu}} \leftarrow \tau_0, \quad \forall(\mathbf{q},\mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$
 3: repeat
 4:     $k \leftarrow 0$; $\mathcal{S}_{\text{trial}} \leftarrow \emptyset$
 5:     for all ants $c$ in parallel do
 6:         $s_{\text{p},c} \leftarrow \emptyset$
 7:         $\mathbf{x}_c \leftarrow \text{random}(\mathcal{X})$
 8:         repeat
 9:             $\mathbf{q}_c \leftarrow \text{quantize}(\mathbf{x}_c)$
10:             $\mathbf{u}_c \sim p_c\{\mathbf{u}|\mathbf{q}_c\} = \frac{\tau^{\alpha}_{\mathbf{q}_c\mathbf{u}}}{\sum_{\mathbf{l}\in\mathcal{U}_{\mathbf{q}_c}} \tau^{\alpha}_{\mathbf{q}_c\mathbf{l}}}, \quad \forall \mathbf{u} \in \mathcal{U}_{\mathbf{q}_c}$
11:             $s_{\text{p},c} \leftarrow s_{\text{p},c} \bigcup \{(\mathbf{q}_c, \mathbf{u}_c)\}$
12:             $\mathbf{x}_c(k+1) \leftarrow \mathbf{f}(\mathbf{x}_c(k), \mathbf{u}_c)$
13:             Local pheromone update:
                $\tau_{\mathbf{q}_c\mathbf{u}_c}(k+1) \leftarrow (1-\gamma)\tau_{\mathbf{q}_c\mathbf{u}_c}(k) + \gamma\tau_0$
14:             $k \leftarrow k+1$
15:         until $k = K_{\max}$
16:         $\mathcal{S}_{\text{trial}} \leftarrow \mathcal{S}_{\text{trial}} \bigcup \{s_{\text{p},c}\}$
17:     end for
18:     Global pheromone update:
        $\tau_{\mathbf{qu}}(\kappa+1) \leftarrow (1-\rho)\tau_{\mathbf{qu}}(\kappa) + \rho \sum_{\substack{s\in\mathcal{S}_{\text{trial}};\\ (\mathbf{q},\mathbf{u})\in s}} J^{-1}(s), \quad \forall(\mathbf{q},\mathbf{u}) : \exists s \in \mathcal{S}_{\text{trial}}; (\mathbf{q},\mathbf{u}) \in s$
19:     $\kappa \leftarrow \kappa+1$
20: until $\kappa = \kappa_{\max}$
Output: $\tau_{\mathbf{qu}}, \quad \forall(\mathbf{q},\mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$
```

3.4 ACL with Fuzzy State Space Partitioning

The number of bins required to accurately capture the dynamics of the original system may become very large even for simple systems with only two state variables. Moreover, the time complexity of ACL grows exponentially with the number of bins, making the algorithm infeasible for realistic systems. In particular, note that for systems with fast dynamics in certain regions of the state space, the sample time needs to be chosen smaller in order to capture the dynamics in these regions accurately. In other regions of the state space where the dynamics of the system are slower, the faster sampling requires a denser quantization, increasing the number of bins. All together, without much prior knowledge of the system dynamics, both

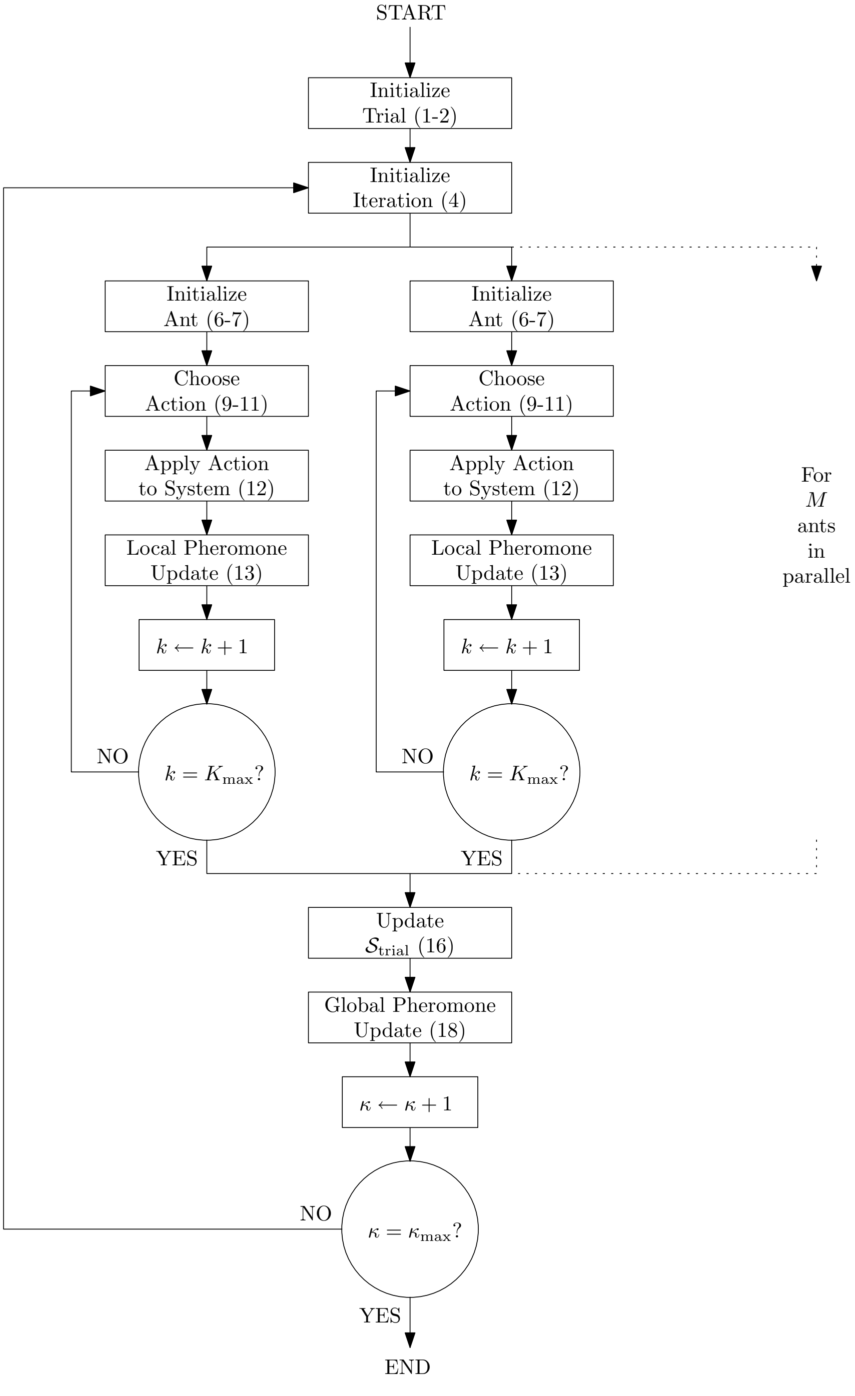


Fig. 2 Flow diagram of the Ant Colony Learning algorithm for optimal control problems. The numbers between parentheses refer to the respective lines of Algorithm 1.

the sampling time and the bin size need to be small enough, resulting in a rapid explosion of the number of bins.

A much better alternative is to approximate the state space by a parametrized function approximator. In that case, there is still a finite number of parameters, but this number can typically be chosen to be much smaller compared to using crisp quantization. The universal function approximator that is used in this chapter is the fuzzy approximator.

3.4.1 Fuzzy State Space Partitioning

With fuzzy approximation, the domain of each state variable is partitioned using membership functions. We define the membership functions for the state variables to be triangular-shaped, such that the membership degrees for any value of the state on the domain always sum up to one. Only the centers of the membership functions have to be stored. An example of such a fuzzy partitioning is given in Figure 3.4.1.

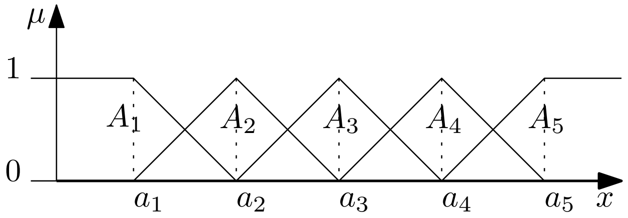


Fig. 3 Membership functions $A_1, \ldots A_5$, with centers $a_1, \ldots, a_5$ on an infinite domain.

Let A_i denote the membership functions for x_1, with a_i their centers for $i = 1, \ldots, N_A$, with N_A the number of membership functions for x_1. Similarly for x_2, denote the membership functions by B_i, with b_i their centers for $i = 1, \ldots, N_B$, with N_B the number of membership functions for x_2. Similarly, the membership functions can be defined for the other state variables in $\mathbf{x}$, but for the sake of notation, the discussion in this chapter limits the number to two, without loss of generality. Note that in the example in Section 4, the order of the system is four.

The membership degree of A_i and B_i are respectively denoted by $\mu_{A_i}(x_1(k))$ and $\mu_{B_i}(x_2(k))$ for a specific value of the state at time k. The degree of fulfillment is computed by multiplying the two membership degrees:

$$\beta_{ij}(\mathbf{x}(k)) = \mu_{A_i}(x_1(k)) \cdot \mu_{B_j}(x_2(k)).$$

Let the vector of all degrees of fulfillment for a certain state at time k be denoted by:

$$
\begin{aligned}
\boldsymbol{\beta}(\mathbf{x}(k)) =& [\beta_{11}(\mathbf{x}(k)) \quad \beta_{12}(\mathbf{x}(k)) \quad \dots \quad \beta_{1N_B}(\mathbf{x}(k)) \\
& \beta_{21}(\mathbf{x}(k)) \quad \beta_{22}(\mathbf{x}(k)) \quad \dots \quad \beta_{2N_B}(\mathbf{x}(k)) \\
& \dots \quad \beta_{N_A N_B}(\mathbf{x}(k))]^{\mathrm{T}},
\end{aligned} \tag{11}
$$

which is a vector containing β_{ij} for all combinations of i and j. Each element will be associated to a vertex in the graph used by Fuzzy ACL. Most of the steps taken from the AS and ACS algorithms for dealing with the $\boldsymbol{\beta}(\mathbf{x}(k))$ vectors from (11) need to be reconsidered. This is the subject of the following section.

3.4.2 Fuzzy ACL Algorithm

In the crisp version of ACL, all combinations of these quantized states for the different state variables corresponded to the nodes in the graph and the arcs corresponded to transitions from one quantized state to another. Because of the quantization, the resulting system was transformed into a stochastic decision problem. However, the pheromones were associated to these arcs as usual. In the fuzzy case, the state space is partitioned by membership functions and the combination of the indices to these membership functions for the different state variables correspond to the nodes in the construction graph. With the fuzzy interpolation, the system remains a deterministic decision problem, but the transition from node to node now does not directly correspond to a state transition. The pheromones are associated to the arcs as usual, but the updating needs to take into account the degree of fulfillment of the associated membership functions.

In Fuzzy ACL, which we have introduced in [12], an ant is not assigned to a certain vertex at a certain time, but to all vertices according to some degree of fulfillment at the same time. For this reason, a pheromone τ_{ij} is now denoted as $\tau_{i\mathbf{u}}$ with i the index of the vertex (i.e. the corresponding element of $\boldsymbol{\beta}$) and $\mathbf{u}$ the action. Similar to the definition of the vector of all degrees of fulfillment in (11), the vector of all pheromones for a certain action $\mathbf{u}$ at time k is denoted as:

$$
\boldsymbol{\tau}_{\mathbf{u}}(k) = \left[\tau_{1\mathbf{u}}(k) \; \tau_{2\mathbf{u}}(k) \; \dots \; \tau_{N_A N_B \mathbf{u}}(k)\right]^{\mathrm{T}}. \tag{12}
$$

Action Selection

The action is chosen randomly according to the probability distribution:

$$
p_c\{\mathbf{u}|\boldsymbol{\beta}_c(k)\}(k) = \sum_{i=1}^{N_A N_B} \beta_{c,i}(k) \frac{\tau_{i,\mathbf{u}}^{\alpha}(k)}{\sum_{\ell \in \mathcal{U}} \tau_{i,\ell}^{\alpha}(k)}. \tag{13}
$$

Note that when $\boldsymbol{\beta}_c$ contains exactly one 1 and for the rest only zeros, this would correspond to the crisp case, where the state is quantized to a set of bins and (13) then reduces to (7).

Local Pheromone Update

The local pheromone update from (3) can be modified to the fuzzy case as follows:

$$\begin{aligned}\boldsymbol{\tau}_{\mathbf{u}} \leftarrow & \boldsymbol{\tau}_{\mathbf{u}}(1-\boldsymbol{\beta}) + ((1-\gamma)\boldsymbol{\tau}_{\mathbf{u}} + \gamma\tau_0)\boldsymbol{\beta} \\ & = \boldsymbol{\tau}_{\mathbf{u}}(1-\gamma\boldsymbol{\beta}) + \tau_0(\gamma\boldsymbol{\beta}),\end{aligned} \tag{14}$$

where all operations are performed element-wise.

As all ants update the pheromone levels associated with the state just visited and the action just taken in parallel, one may wonder whether or not the order in which the updates are done matters, when the algorithm is executed on a standard CPU, where all operations are done in series. With crisp quantization, the ants may indeed sometimes visit the same state and with fuzzy quantization, the ants may very well share some of the membership functions with a membership degree larger than zero. We will show that in both cases, the order of updates in series does not influence the final value of the pheromones after the joint update. In the crisp case, the local pheromone update from (8) may be rewritten as follows:

$$\begin{aligned}\tau_{\mathbf{qu}}^{(1)} \leftarrow & (1-\gamma)\tau_{\mathbf{qu}} + \gamma\tau_0 \\ & = (1-\gamma)(\tau_{\mathbf{qu}} - \tau_0) + \tau_0.\end{aligned}$$

Now, when a second ant updates the same pheromone level $\tau_{\mathbf{qu}}$, the pheromone level becomes:

$$\begin{aligned}\tau_{\mathbf{qu}}^{(2)} \leftarrow & (1-\gamma)(\tau_{\mathbf{qu}}^{(1)} - \tau_0) + \tau_0 \\ & = (1-\gamma)^2(\tau_{\mathbf{qu}} - \tau_0) + \tau_0.\end{aligned}$$

After n updates, the pheromone level is:

$$\tau_{\mathbf{qu}}^{(n)} \leftarrow (1-\gamma)^n(\tau_{\mathbf{qu}} - \tau_0) + \tau_0, \tag{15}$$

which shows that the order of the update is of no influence to the final value of the pheromone level.

For the fuzzy case a similar derivation can be made. In general, after all the ants have performed the update, the pheromone vector is:

$$\boldsymbol{\tau}_{\mathbf{u}} \leftarrow \left(\prod_{c=1}^{M}(1-\gamma\boldsymbol{\beta}_c)\right)(\boldsymbol{\tau}_{\mathbf{u}} - \tau_0) + \tau_0, \tag{16}$$

where again all operations are performed element-wise. This result also reveals that the final values of the pheromones are invariant with respect to the order of updates. Furthermore, also note that when $\boldsymbol{\beta}_c$ contains exactly one 1 and for the rest only zeros, corresponding to the crisp case, the fuzzy local pheromone update from either (14) or (16) reduces to the crisp case in respectively (8) or (15).

Global Pheromone Update

In order to derive a fuzzy representation of the global pheromone update step it is convenient to rewrite (10) using indicator functions:

$$\tau_{\mathbf{qu}}(\kappa+1) \leftarrow \left\{(1-\rho)\tau_{\mathbf{qu}}(\kappa) + \rho \sum_{s \in \mathcal{S}_{\text{upd}}} J^{-1}(s)\mathcal{I}_{\mathbf{qu},s}\right\}\mathcal{I}_{\mathbf{qu},\mathcal{S}_{\text{upd}}} + \tau_{\mathbf{qu}}(\kappa)(1-\mathcal{I}_{\mathbf{qu},\mathcal{S}_{\text{upd}}}). \tag{17}$$

In (17), $\mathcal{I}_{\mathbf{qu},\mathcal{S}_{\text{upd}}}$ is an indicator function that can take values from the set $\{0,1\}$ and is equal to 1 when $(\mathbf{q},\mathbf{u})$ is an element of a solution belonging to the set $\mathcal{S}_{\text{upd}}$ and 0 otherwise. Similarly, $\mathcal{I}_{\mathbf{qu},s}$ is an indicator function as well, and it is equal to 1 when $(\mathbf{q},\mathbf{u})$ is an element of the solutions s and 0 otherwise.

Now, by using the vector of pheromones from (12), we introduce the vector indicator function $\mathcal{I}_{\mathbf{u},s}$, being a vector of the same size as $\boldsymbol{\tau}_u$, with elements from the set $\{0,1\}$ and for each state indicating whether it is an element of s. A similar vector indicator function $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}$ is introduced as well. Using these notations, we can write (17) for all states q together as:

$$\boldsymbol{\tau}_{\mathbf{u}}(\kappa+1) \leftarrow \left\{(1-\rho)\boldsymbol{\tau}_{\mathbf{u}}(\kappa) + \rho \sum_{s \in \mathcal{S}_{\text{upd}}} J^{-1}(s)\mathcal{I}_{\mathbf{u},s}\right\}\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{upd}}} + \boldsymbol{\tau}_{\mathbf{u}}(\kappa)(1-\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{upd}}}), \tag{18}$$

where all multiplications are performed element-wise.

The most basic vector indicator function is $\mathcal{I}_{\mathbf{u},s^{(i)}}$, which has only one element equal to 1, namely the one for $(\mathbf{q},\mathbf{u}) = s^{(i)}$. Recall that a solution $s = \{s^{(1)}, s^{(2)}, \ldots, s^{(N_{\text{s}})}\}$ is an ordered set of solution components $s^{(i)} = (\mathbf{q}_i, \mathbf{u}_i)$. Now, $\mathcal{I}_{\mathbf{u},s}$ can be created by taking the union of all $\mathcal{I}_{\mathbf{u},s^{(i)}}$:

$$\mathcal{I}_{\mathbf{u},s} = \mathcal{I}_{\mathbf{u},s^{(1)}} \cup \mathcal{I}_{\mathbf{u},s^{(2)}} \cup \ldots \cup \mathcal{I}_{\mathbf{u},s^{(N_{\text{s}})}} = \bigcup_{i=1}^{N_{\text{s}}} \mathcal{I}_{\mathbf{u},s^{(i)}}, \tag{19}$$

where we define the union of indicator functions in the light of the fuzzy representation of the state by a fuzzy union set operator, such as:

$$(A \cup B)(x) = \max[\mu_A(x), \mu_B(x)]. \tag{20}$$

In this operator, where A and B are membership functions, x a variable and $\mu_A(x)$ the degree to which x belongs to A. Note that when A maps x to a crisp domain $\{0,1\}$, the union operator is still valid. Similar to (19), if we denote the set of solutions as the ordered set of solutions $\mathcal{S}_{\text{upd}} = \{s_1, s_2, \ldots, s_{N_{\text{S}}}\}$, $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}$ can be created by taking the union of all $\mathcal{I}_{\mathbf{u},s}$:

$$\mathcal{I}_{\mathbf{u},\mathcal{S}_{\mathrm{trial}}} = \mathcal{I}_{\mathbf{u},s_1} \cup \mathcal{I}_{\mathbf{u},s_2} \cup \ldots \cup \mathcal{I}_{\mathbf{u},s_{N_{\mathrm{S}}}} = \bigcup_{i=1}^{N_{\mathrm{S}}} \mathcal{I}_{\mathbf{u},s_i}. \tag{21}$$

In fact, $\mathcal{I}_{\mathbf{u},s_i}$ can be regarded as a representation of the state-action pair $(\mathbf{q}_i, \mathbf{u}_i)$. In computer code, $\mathcal{I}_{\mathbf{u},s_i}$ may be represented by a matrix of dimensions $|\mathcal{Q}| \cdot |\mathcal{U}|$ containing only one element equal to 1 and the others zero[3].

In order to generalize the global pheromone update step to the fuzzy case, realize that $\boldsymbol{\beta}(\mathbf{x}(k))$ from (11) can be seen as a generalized (fuzzy, or graded) indicator function $\mathcal{I}_{\mathbf{u},s^{(i)}}$, if combined with an action $\mathbf{u}$. The elements of this generalized indicator function take values in the range $[0,1]$, for which the extremes 0 and 1 form the special case of the original (crisp) indicator function. We can thus use the state-action pair where the state has the fuzzy representation of (11) directly to compose $\mathcal{I}_{\mathbf{u},s^{(i)}}$. Based on this, we can compose $\mathcal{I}_{\mathbf{u},s}$ and $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\mathrm{trial}}}$ in the same way as in (19) and (21) respectively. With the union operator from (20) and the definition of the various indicator functions, the global pheromone update rule from (18) can be used in both the crisp and fuzzy variant of ACL.

Note that we need more memory to store the fuzzy representation of the state compared to its crisp representation. With pair-wise overlapping normalized membership functions, like the ones shown in Figure 3.4.1, at most 2^d elements are nonzero, with d the dimension of the state space. This means an exponentially growth in memory requirements for increasing state space dimension, but also that it is independent of the number of membership functions used for representing the state space.

As explained in Section 3.2, for optimal control problems, the appropriate update rule is to use all solutions by all ants in the trial $\mathcal{S}_{\mathrm{trial}}$. In the fuzzy case, the solutions $s \in \mathcal{S}_{\mathrm{trial}}$ consist of sequences of states and actions, and the states can be fuzzified so that they are represented by sequences of vectors of degrees of fulfillment $\boldsymbol{\beta}$. Instead of one pheromone level, in the fuzzy case a set of pheromone levels are updated to a certain degree. It can be easily seen that as this update process is just a series of pheromone *deposits*, the final value of the pheromone levels relates to the sum of these deposits and is invariant with respect to the order of these deposits. This is also the case for this step in Crisp ACL.

Regarding the terminal condition for the ants, with the fuzzy implementation, none of the vertices can be identified as being the terminal vertex. Rather one has to define a set of membership functions that can be used to determine to what degree the goal state has been reached. These membership functions can be used to express the linguistic fuzzy term of the state being *close to the goal*. Specifically, this is satisfied when the membership degree of the state to the membership function with its core equal to the goal state is larger than 0.5. If this has been satisfied, the ant has terminated its trial.

[3] In MATLAB this may conveniently be represented by a sparse matrix structure, rendering the indicator-representation useful for generalizing the global pheromone update rule and while still being a memory efficient representation of the state-action pair.

3.5 Parameter Settings

Some of the parameters in both Crisp and Fuzzy ACL are similarly initialized as in the ACS. The global and local pheromone trail decay factors are set to a preferably small value, respectively $\rho \in (0,1)$ and $\gamma \in (0,1)$. There will be no heuristic parameter associated to the arcs in the construction graph, so only an exponential weighting factor for the pheromone trail $\alpha > 0$ has to be chosen. Increasing α leads to more biased decisions towards the one corresponding to the highest pheromone level. Choosing $\alpha = 2$ or 3 appears to be an appropriate choice. Furthermore, the control parameters for the algorithm need to be chosen, such as the maximum number of iterations per trial, $K_{\max}$, and the maximum number of trials, $T_{\max}$. The latter one can be set to, e.g., 100 trials, where the former one depends on the sample time T_{s} and a guess of the time needed to get from the initial state to the goal optimally, T_{guess}. A good choice for $K_{\max}$ would be to take 10 times the expected number of iterations, $K_{\max} = 10T_{\mathrm{guess}}/T_{\mathrm{s}}$. Specific to ACL, the number and shape of the membership functions, or the number of quantization bins, and their spacing over the state domain need to be determined. Furthermore, the pheromones are initialized as $\tau_{i\mathbf{u}} = \tau_0$ for all $(i, \mathbf{u})$ in Fuzzy ACL and as $\tau_{\mathbf{qu}} = \tau_0$ for all $(\mathbf{q}, \mathbf{u})$ in Crisp ACL and where τ_0 is a small, positive value. Finally the number of ants M must be chosen large enough such that the complete state space can be visited frequently enough.

3.6 Relation to Reinforcement Learning

In reinforcement learning [26], the agent interacts in a step-wise manner with the system, experiencing state transitions and rewards. These rewards are a measure of the short-time quality of the taken decision and the objective for the agent is to create a state-action mapping (the policy) that maximizes the (discounted) sum of these rewards. Particularly in Monte Carlo learning, the agent interacts with the system and stores the states that it visited and the actions that it took in those states until it reaches some goal state. Now, its trial ends and the agent evaluates the sequence of state-action pairs over some cost function and based on that updates a value function. This value function basically stores the information about the quality of being in a certain state (or in the Q-learning [30] and SARSA [31] variants, the quality of choosing a particular action in a certain state). Repeating this procedure of interacting with the system and updating its value function will eventually result in the value function converging to the optimal value function, which corresponds to the solution of the control problem. It is said that the control policy has been *learned*, because it has been automatically derived purely by interaction with the system.

This procedure shows strong similarities with the procedure of ACL. The main difference is that rather than a single agent, in ACL there is a multitude of agents (called ants) all interacting with the system in parallel and all contributing to the update of the value function (called the set of pheromone levels). For exactly the same reason as with reinforcement learning, the procedure in ACL enables the ants to *learn* the optimal control policy.

However, as all ants work in parallel, the interaction of all these ants with a single real world system is problematic. You would require multiple copies of the system in order to keep up this parallelism. In simulation, this is not an issue. If there is a model of the real world system available, making many copies of this model in software is very cheap. Nevertheless, it is not completely correct to call ACL for this reason a *model-based* learning algorithm. As the model may be a blackbox model, with the details unknown to the ants, it is purely a way to benefit from the parallelism of ACL.

Because of the parallelism of the ants, ACL could be called a multi-agent learning algorithm. However, this must not be confused with multi-agent reinforcement learning, as this stands for the setting in which multiple reinforcement learning agents all act in a single environment according to their own objective and in order for each of them to behave optimally, they need to take into account the behavior of the other agents in the environment. Note that in ACL, the ants do not need to consider, or even notice, the existence of the other ants. All they do is benefit from each other's findings.

4 Example: Navigation with Variable Damping

This section presents an example application of both Crisp and Fuzzy ACL to a continuous-state dynamic system. The dynamic system under consideration is a simulated two-dimensional (2D) navigation problem and similar to the one described in [24]. Note that it is not our purpose to demonstrate the superiority of ACL over any other method for this specific problem. Rather we want to demonstrate the functioning of the algorithm and compare the results for both its versions.

4.1 Problem Formulation

A vehicle, modeled as a point-mass of 1 kg, has to be steered to the origin of a two-dimensional surface from any given initial position in an optimal manner. The vehicle experiences a damping that varies non-linearly over the surface. The state of the vehicle is defined as $\mathbf{x} = \begin{bmatrix} c_1 \; v_1 \; c_2 \; v_2 \end{bmatrix}^{\mathrm{T}}$, with c_1, c_2 and v_1, v_2 the position and velocity in the direction of each of the two principal axes respectively. The control input to the system $\mathbf{u} = \begin{bmatrix} u_1 \; u_2 \end{bmatrix}^{\mathrm{T}}$ is a two-dimensional force. The dynamics are:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -b(c_1, c_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -b(c_1, c_2) \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}(t),$$

where the damping $b(c_1, c_2)$ in the experiments is modeled by an affine sum of two Gaussian functions:

$$b(c_1, c_2) = b_0 + \sum_{i=1}^{2} b_i \exp\left[-\sum_{j=1}^{2} \frac{(c_j - m_{j,i})^2}{\sigma_{j,i}^2} \right],$$

for which the chosen values of the parameters are $b_0 = 0.5$, $b_1 = b_2 = 8$, and $m_{1,1} = 0$, $m_{2,1} = -2.3$, $\sigma_{1,1} = 2.5$, $\sigma_{2,1} = 1.5$ for the first Gaussian, and $m_{1,2} = 4.7$, $m_{2,2} = 1$, $\sigma_{1,2} = 1.5$, $\sigma_{2,2} = 2$ for the second Gaussian. The damping profile can be seen in, e.g., Figure 7, where darker shading means more damping.

4.2 State Space Partitioning and Parameters

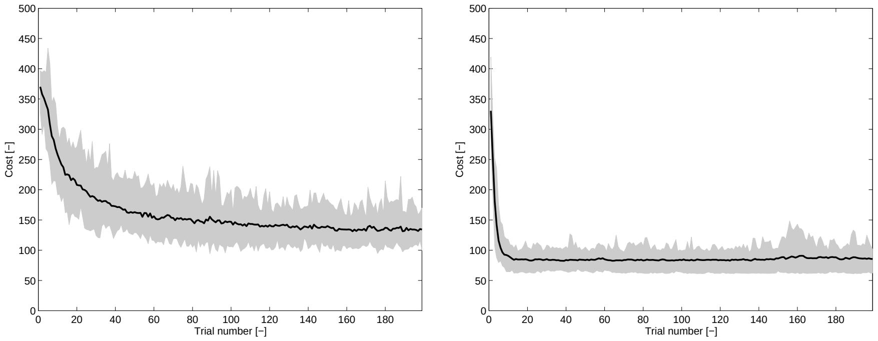
The partitioning of the state space is as follows:

- For the position c_1: $\{-5, -3.75, -2, -0.3, 0, 0.3, 2, 2.45, 3.5, 3.75, 4.7, 5\}$ and for c_2: $\{-5, -4.55, -3.5, -2.3, -2, -1.1, -0.6, -0.3, 0, 0.3, 1, 2.6, 4, 5\}$.
- For the velocity in both dimensions v_1, v_2: $\{-2, 0, 2\}$.

This particular partitioning of the position space is composed of a baseline grid $\{-5 - 0.300.35\}$ and adding to it, extra grid lines inside each Gaussian damping region. This partitioning is identical to what is used in [24]. In the crisp case, the crossings in the grid are the centers of the quantization bins, where in the fuzzy case, these are the centers of the triangular membership functions. The action set contains of 9 actions, namely the cross-product of the sets $\{-1, 0, 1\}$ for both dimensions. The local and global pheromone decay factors are respectively $\gamma = 0.01$ and $\rho = 0.1$. Furthermore, $\alpha = 3$ and the number of ants is 200. The sampling time is $T_s = 0.2$ and the ants are randomly initialized over the complete state space at the start of each trial. An ant terminates its trial when its position and velocity in both dimensions are within a bound of ± 0.25 and ± 0.05 from the goal respectively. Each experiment is carried out thirty times. The quadratic cost function from (5) is used with the matrices $\mathbf{Q} = \mathrm{diag}(0.2, 0.1, 0.2, 0.1)$ and $\mathbf{R} = 0$. The cost function thus only takes into account the deviation of the state to the goal. We will run the fuzzy Q-iteration algorithm from [24] for this problem with the same state space partitioning in order to derive the optimal policy to which we can compare the policies derived by both versions of the ACL algorithm.

4.3 Results

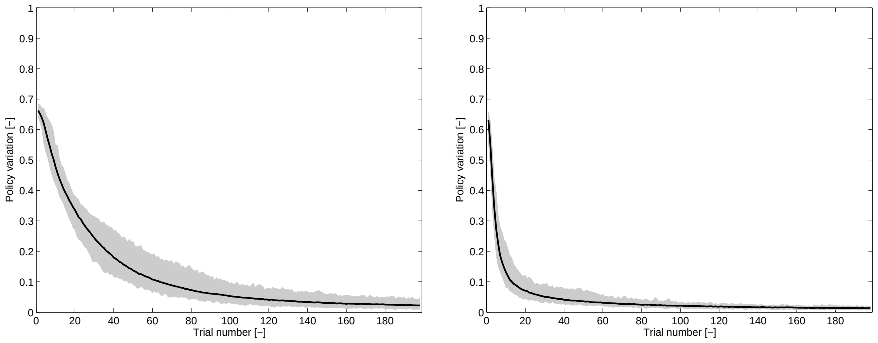
The first performance measure considered represents the cost of the control policy as a function of the number of trials. The cost of the control policy is measured as the average cost of a set of trajectories resulting from simulating the system controlled by the policy and starting from a set of 100 pre-defined initial states, uniformly distributed over the state space. Figure 4 shows these plots. As said earlier, each experiment is carried out thirty times. The black line in the plots is the average cost over these thirty experiments and the gray area represents the range of costs for the thirty experiments. From Figure 4 it can be concluded that the Crisp ACL algorithm does not converge as fast and smoothly compared to Fuzzy ACL and that it converges to a larger average cost. Furthermore, the gray region for Crisp ACL is larger than that of Fuzzy ACL, meaning that there is a larger variety of policies that the algorithm converges to.



(a) Crisp ACL (b) Fuzzy ACL

Fig. 4 The cost of the control policy as a function of the number of trials passed since the start of the experiment. The black line in each plot is the average cost over thirty experiments and the gray area represents the range of costs for these experiments.

The convergence of the algorithm can also be measured by the fraction of quantized states (crisp version), or cores of the membership functions (fuzzy version) for which the policy has changed at the end of a trial. This measure will be called the *policy variation*. The policy variation as a function of the trials for both Crisp and Fuzzy ACL is depicted in Figure 5. It shows that for both ACL versions, the policy variation never really becomes completely zero and confirms that Crisp ACL converges slower compared to Fuzzy ACL. It also provides the insight that although for Crisp ACL the pheromone levels do not converge to the same value at every run of the algorithm, they do all converge in about the same way in the sense of policy variation. This is even more true for Fuzzy ACL, where the gray area around the average policy variation almost completely disappears for an increasing number of trials. The observation that the policy variation never becomes completely zero indicates that there are some states for which either action results in nearly the same cost.



(a) Crisp ACL

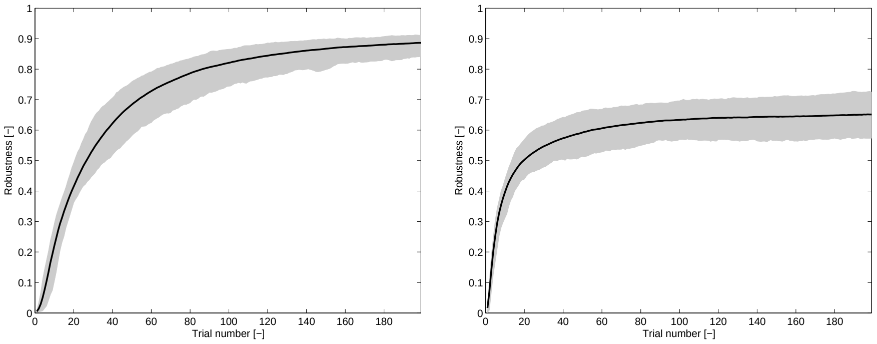
(b) Fuzzy ACL

Fig. 5 Convergence of the algorithm in terms of the fraction of states (cores of the membership functions, or centers of the quantization bins) for which the policy changed at the end of a trial. The black line in each plot is the average policy variation over thirty experiments and the gray area represents the range of policy variation for these experiments.

A third performance measure that is considered represents the robustness of the control policy. For each state, there is a certain number of actions to choose from. If the current policy indicates that a certain action is the best, it does not say how much better it is compared to the second-best action. If it is only a little better, a small update of the pheromone level for that other action in this state may switch the policy for this state. In that case, the policy is not considered to be very robust. The robustness for a certain state is defined as follows:

$$\begin{aligned}\mathbf{u}_{\max1}(\mathbf{q}) &= \arg\max_{\mathbf{u}\in\mathcal{U}}(\tau_{\mathbf{qu}}),\\ \mathbf{u}_{\max2}(\mathbf{q}) &= \arg\max_{\mathbf{u}\in\mathcal{U}\setminus\mathbf{u}_{\max1}(\mathbf{q})}(\tau_{\mathbf{qu}}),\\ \mathrm{robustness}(\mathbf{q}) &= \frac{\tau^{\alpha}_{\mathbf{q}\mathbf{u}_{\max1}(\mathbf{q})} - \tau^{\alpha}_{\mathbf{q}\mathbf{u}_{\max2}(\mathbf{q})}}{\tau^{\alpha}_{\mathbf{q}\mathbf{u}_{\max1}(\mathbf{q})}},\end{aligned}$$

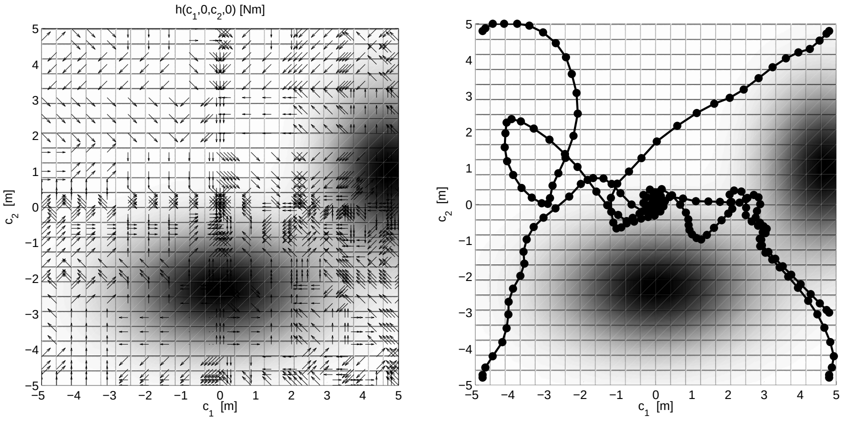
where α is the same as the α from the Boltzmann action selection rule ((13) and (7)). The robustness of the policy is the average robustness over all states. Figure 6 shows the evolution of the average robustness during the experiments. The figure shows that the robustness increases with the number of trials, explaining the decrease in the policy variation from Figure 5. The fact that it does not converge to one explains that there remains a probability larger than zero for the policy to change. This suggests that ACL algorithms are potentially capable of adapting the policy to changes in the cost function, due to changing system dynamics, or a change of the goal state.



(a) Crisp ACL (b) Fuzzy ACL

Fig. 6 Evolution of the robustness of the control policy as a function of the number of trials. The black line in each plot is the average robustness over thirty experiments and the gray area represents the range of the robustness for these experiments.

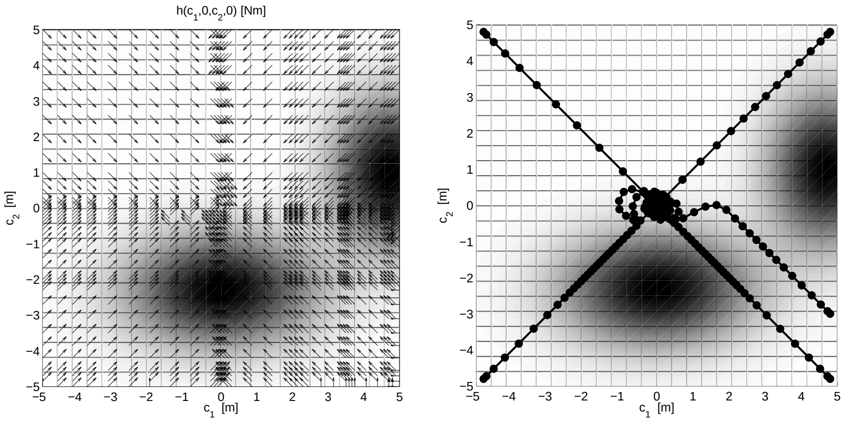
Finally, we present the policies and the behavior of a simulated vehicle controlled by these policies for both Crisp and Fuzzy ACL. In the case of Crisp ACL, Figure 7(a) depicts a slice of the resulted policy for zero velocity. It shows the mapping of the positions in both dimensions to the input. Figure 7(b) presents the trajectories of the vehicle for various initial positions and zero initial velocity. In both



(a) Crisp ACL: policy (b) Crisp ACL: simulation

Fig. 7 The left figure presents a slice of the best resulting policy for zero velocity obtained by Crisp ACL in one of the thirty runs. It shows the control input for a fine grid of positions. The multi-Gaussian damping profile is shown, where darker shades represent regions of more damping. The figure on the right shows the trajectories of the vehicle under this policy for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

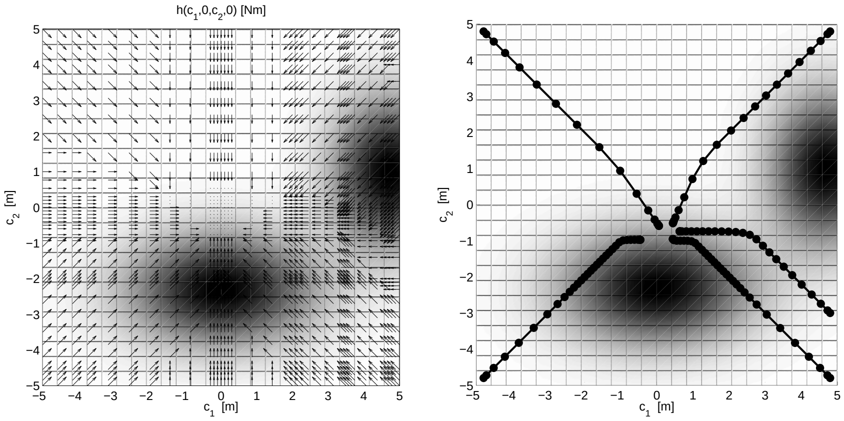
figures, the mapping is shown for a grid three times finer than the grid spanned by the cores of the membership functions. For the crisp case, the states in between the centers of the partition bins are quantized to the nearest center. Figure 8 presents the results for Fuzzy ACL. Comparing these results with those from the Crisp ACL algorithm in Figure 7, it shows that for Crisp ACL, the trajectories of the vehicle go widely around the regions of stronger damping, in one case (starting from the top-left corner) even clearly taking a suboptimal path to the goal. For Fuzzy ACL, the trajectories are very smooth and seem to be more optimal, however the vehicle does not avoid the regions of stronger damping much. The policy of Fuzzy ACL is also much more regular compared to the one obtained by Crisp ACL.



(a) Fuzzy ACL: policy (b) Fuzzy ACL: simulation

Fig. 8 The left figure presents a slice of the best resulting policy for zero velocity obtained by Fuzzy ACL in one of the thirty runs. It shows the control input for a fine grid of positions. The multi-Gaussian damping profile is shown, where darker shades represent regions of more damping. The figure on the right shows the trajectories of the vehicle under this policy for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

In order to verify the optimality of the resulting policies, we also present the optimal policy and trajectories obtained by the fuzzy Q-iteration algorithm from [24]. This algorithm is a model-based reinforcement learning method developed for continuous state spaces and is guaranteed to converge to the optimal policy on the partitioning grid. Figure 9 present the results for fuzzy Q-iteration. It can be seen that the optimal policy from fuzzy Q-iteration is very similar to the one obtained by Fuzzy ACL, but that it manages to avoid the regions of stronger damping somewhat better. Even with the optimal policy, the regions of stronger damping are not completely correctly avoided. Especially regarding the policy around the lower damping region, where it steers the vehicle towards an even stronger damped part of the state space. The reason for this is the particular choice of the cost function in these experiments. The results in [24] show that for a discontinuous cost function, where



(a) Fuzzy Q-iteration: Policy (b) Fuzzy Q-iteration: simulation

Fig. 9 The baseline optimal policy derived by the fuzzy Q-iteration algorithm. The left figure presents a slice of the policy for zero velocity. The figure on the right shows the trajectories of the vehicle under this policy for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

reaching the goal results in a sudden and steep decrease in cost, the resulting policy avoids the highly damped regions much better. Also note that in those experiments, the policy interpolates between the actions resulting in a much smoother change of the policy between the centers of the membership functions. Theoretically, Fuzzy ACL also allows for interpolation of the action set, but this is outside the scope of this chapter. Also, the ACL framework allows for different cost functions than the quadratic cost function used in this example. Designing the cost function carefully is very important for any optimization or control problem and directly influences the optimal policy. It is however not the purpose of this chapter to find the best cost function for this particular control problem.

5 Conclusions and Future Work

This chapter has discussed Ant Colony Learning (ACL), a novel method for the design of optimal control policies for continuous-time, continuous-state dynamic systems based on Ant Colony Optimization (ACO). The partitioning of the state space is a crucial aspect to ACL and two versions have been presented with respect to this. In Crisp ACL, the state space is partitioned using bins, such that each value of the state maps to exactly one bin. Fuzzy ACL, on the other hand, uses a partitioning of the state space with triangular membership functions. In this case, each value of the state maps to the membership functions to a certain membership degree. Both ACL algorithms are based on the combination of the Ant System and the Ant Colony

System, but have some distinctive characteristics. The action selection step does not include an explicit probability of choosing the action associated with the highest value of the pheromone matrix in a given state, but only uses a random-proportional rule. In this rule, the heuristic variable, typically present in ACO algorithms, is left out as prior knowledge can also be incorporated through an initial choice of the pheromone levels. The absence of the heuristic variable makes ACL more straightforward to apply. Another particular characteristic of our algorithm is that it uses the complete set of solutions of the ants in each trial to update the pheromone matrix, whereas most ACO algorithms typically use some form of elitism, selecting only one of the solutions from the set of ants to perform the global pheromone update. In a control setting, this is not possible, as ants initialized closer to the goal will have experienced a lower cost than ants initialized further away from the goal, but this does not mean that they must be favored in the update of the pheromone matrix.

The applicability of ACL to optimal control problems with continuous-valued states is outlined and demonstrated on the non-linear control problem of two-dimensional navigation with variable damping. The results show convergence of both the crisp and fuzzy version of the algorithm to suboptimal policies. However, Fuzzy ACL converged much faster and the cost of its resulting policy did not change as much over repetitive runs of the algorithm compared to Crisp ACL. It also converged to a more optimal policy than Crisp ACL. We have presented the additional performance measures of policy variation and robustness in order to study the convergence of the pheromone levels in relation to the policy. These measures showed that ACL converges as a function of the number of trials to an increasingly robust control policy meaning that a small change in the pheromone levels will not result in a large change in the policy. Compared to the optimal policy derived from fuzzy Q-iteration, the policy from Fuzzy ACL appeared to be close to optimal. It was noted that the policy could be improved further by choosing a different cost function. Interpolation of the action space is another way to get a smoother transition of the policy between the centers of the state space partitioning. Both these issues were not considered in more detail in this chapter and we recommend to look further into this in future research.

The ACL algorithm currently did not incorporate prior knowledge of the system, or of the optimal control policy. However, sometimes prior knowledge of such kind is available and using it could be a way to speed up the convergence of the algorithm and to increase the optimality of the derived policy. Future research must focus on this issue. Comparing ACL to other related methods, like for instance Q-learning, is also part of our plans for future research. Finally, future research must focus on a more theoretical analysis of the algorithm, such as a formal convergence proof.

References

1. M. Dorigo and C. Blum, “Ant colony optimization theory: a survey,” *Theoretical Computer Science*, vol. 344, no. 2-3, pp. 243–278, November 2005.

2. A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, F. J. Varela and P. Bourgine, Eds. Cambridge, MA: MIT Press, 1992, pp. 134–142.
3. M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: The MIT Press, 2004.
4. K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 33, no. 5, pp. 560–572, September 2003.
5. R. H. Huang and C. L. Yang, "Ant colony system for job shop scheduling with time windows," *International Journal of Advanced Manufacturing Technology*, vol. 39, no. 1-2, pp. 151–157, 2008.
6. K. Alaykran, O. Engin, and A. Dyen, "Using ant colony optimization to solve hybrid flow shop scheduling problems," *International Journal of Advanced Manufacturing Technology*, vol. 35, no. 5-6, pp. 541–550, 2007.
7. X. Fan, X. Luo, S. Yi, S. Yang, and H. Zhang, "Optimal path planning for mobile robots based on intensified ant colony optimization algorithm," in *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP 2003)*, Changsha, Hunan, China, October 2003, pp. 131–136.
8. J. Wang, E. Osagie, P. Thulasiraman, and R. K. Thulasiram, "HOPNET: A hybrid ant colony optimization routing algorithm formobile ad hoc network," *Ad Hoc Networks*, vol. 7, no. 4, pp. 690–705, 2009.
9. A. H. Purnamadjaja and R. A. Russell, "Pheromone communication in a robot swarm: necrophoric bee behaviour and its replication," *Robotica*, vol. 23, no. 6, pp. 731–742, 2005.
10. B. Fox, W. Xiang, and H. P. Lee, "Industrial applications of the ant colony optimization algorithm," *International Journal of Advanced Manufacturing Technology*, vol. 31, no. 7-8, pp. 805–814, 2007.
11. L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "Metaheuristics in stochastic combinatorial optimization: a survey." IDSIA, Manno, Switzerland, Tech. Rep. 08, March 2006.
12. J. M. van Ast, R. Babuška, and B. De Schutter, "Novel ant colony optimization approach to optimal control," *International Journal of Intelligent Computing and Cybernetics*, vol. 2, no. 3, pp. 414 – 434, 2009.
13. ——, "Fuzzy ant colony optimization for optimal control," in *Proceedings of the American Control Conference (ACC 2009)*, no. 1003–1008, Saint Louis, MO, USA, June 2009.
14. K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training," *Neural Computing & Applications*, vol. 16, no. 3, pp. 235–247, May 2007.
15. K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008.
16. G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," in *Selected Papers from AISB Workshop on Evolutionary Computing*, ser. Lecture Notes in Computer Science, T. Fogarty, Ed., vol. 993. London, UK: Springer-Verlag, April 1995, pp. 25–39.
17. S. Tsutsui, M. Pelikan, and A. Ghosh, "Performance of aggregation pheromone system on unimodal and multimodal problems," in *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, September 2005, pp. 880–887.
18. P. Korosec, J. Silc, K. Oblak, and F. Kosel, "The differential ant-stigmergy algorithm: an experimental evaluation and a real-world application," in *Proceedings of the 2007 Congress on Evolutionary Computation (CEC 2007)*, September 2007, pp. 157–164.
19. M. Birattari, G. D. Caro, and M. Dorigo, "Toward the formal foundation of Ant Programming," in *Proceedings of the International Workshop on Ant Algorithms (ANTS 2002)*. Brussels, Belgium: Springer-Verlag, September 2002, pp. 188–201.

20. L. M. Gambardella and M. Dorigo, “Ant-Q: A reinforcement learning approach to the traveling salesman problem,” in *Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning*, A. Prieditis and S. Russell, Eds. San Francisco, CA: Morgan Kaufmann Publishers, 1995, pp. 252–260.
21. J. Casillas, O. Cordn, and F. Herrera, “Learning fuzzy rule-based systems using ant colony optimization algorithms,” in *Proceedings of the ANTS’2000. From Ant Colonies to Artificial Ants: Second Interantional Workshop on Ant Algorithms. Brussels (Belgium)*, September 2000, pp. 13–21.
22. B. Zhao and S. Li, “Design of a fuzzy logic controller by ant colony algorithm with application to an inverted pendulum system,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Taipei, Taiwan, October 2006, pp. 3790–3794.
23. W. Zhu, J. Chen, and B. Zhu, “Optimal design of fuzzy controller based on ant colony algorithms,” in *Proceedings of the IEEE International Conference on Mechatronics and Automation*, Luoyang, China, June 2006, pp. 1603–1607.
24. L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, “Continuous-state reinforcement learning with fuzzy approximation,” in *Adaptive Agents and Multi-Agent Systems III*, ser. Lecture Notes in Computer Science, K. Tuyls, A. Nowé, Z. Guessoum, and D. Kudenko, Eds. Springer, 2008, vol. 4865, pp. 27–43.
25. M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 29–41, 1996.
26. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
27. M. Dorigo and L. Gambardella, “Ant Colony System: a cooperative learning approach to the traveling salesman problem,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.
28. T. Stützle and U. Hoos, “MAX MIN Ant System,” *Journal of Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.
29. K. J. Åström and B. Wittenmark, *Computer Controlled Systems—Theory and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, January 1990.
30. C. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
31. G. A. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” Cambridge University, Tech. Rep. CUED/F-INFENG/TR166, 1994.