JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 7, August 2010

Reinforcement Learning of a Morphing Airfoil-Policy and Discrete Learning Analysis

Amanda Lampton*
Systems Technology, Inc., Hawthorne, California 90250

and

Adam Niksch† and John Valasek‡
Texas A&M University, College Station, Texas 77843-3141

DOI: 10.2514/1.48057

Casting the problem of morphing a microair vehicle as a reinforcement-learning problem to achieve desired tasks or performance is a candidate approach for handling many of the unique challenges associated with such small aircraft. This paper presents an early stage in the development of learning how and when to morph a micro air vehicle by developing an episodic unsupervised learning algorithm using the Q-learning method to learn the shape and shape change policy of a single morphing airfoil. Reinforcement is addressed by reward functions based on airfoil properties, such as lift coefficient, representing desired performance for specified flight conditions. The reinforcement learning as it is applied to morphing is integrated with a computational model of an airfoil. The methodology is demonstrated with numerical examples of an NACA type airfoil that autonomously morphs in two degrees of freedom, thickness and camber, to a shape that corresponds to specified goal requirements. Because of the continuous nature of the thickness and camber of the airfoil, this paper addresses the convergence of the learning algorithm given several discretizations. Convergence is also analyzed with three candidate policies: 1) a fully random exploration policy, 2) a policy annealing from random exploration to exploitation, and 3) an annealing discount factor in addition to the annealing policy. The results presented in this paper show the inherent differences in the learned action-value function when the state-space discretization, policy, and learning parameters differ. It was found that a policy annealing from fully *explorative* to almost fully *exploitative* yielded the highest rate of convergence as compared to the other policies. Also, the coarsest discretization of the state-space resulted in convergence of the action-value function in as little as 200 episodes.

Nomenclature

A	plane of the airfoil
$A(s_t)$	set of actions available in state s_t
a	action

Received 16 November 2009; accepted for publication 17 June 2010.

* Staff Engineer, Research, AIAA Member, alampton@systemstech.com
† Graduate Research Assistant, Vehicle Systems & Control Laboratory, Aerospace Engineering Department, AIAA Student Member, adam45611@tamu.edu
‡ Associate Professor and Director, Vehicle Systems & Control Laboratory, Aerospace Engineering Department, AIAA Associate Fellow, valasek@tamu.edu

h	distance between vertices
L	length
N	number of state-action pairs
N_V	number of vertices
P	probability
$Q^{\pi}(s, a)$	action-value function for policy π
R	reward
r	discount rate
r_t	reward at time t
S	set of possible states for reinforcement learning
s	state
$V^{\pi}(s)$	state value function for policy π
X	vertex
x	axis

Greek

α	learning rate
γ	discount factor
ε	greedy policy parameter
π	policy

Subscript

i	index
j	index
t	time
1	axis index
2	axis index

Superscript

I	index
J	index
$*$	optimal

I. Introduction

INTERACTING with our environment is one of the fundamental ways in which we learn. With no explicit teacher and only sensory inputs, we can learn much information about cause and effect. We seek to learn and understand how the environment responds to our actions. Reinforcement learning, a machine-learning method, is a computational approach to learning from interaction. The reinforcement-learning problem seeks to learn what to do so as to maximize the numerical reward. Reinforcement learning, in effect, learns how to map situations, or states, to actions. The learner or agent does this by exploring its environment and discovering which actions yield the most reward.

The goal of an algorithm used to solve a reinforcement-learning problem is to converge to a usable policy that will tell the agent how to transition about the state-space to the goal. Problems that arise often involve how the problem in question is cast as a reinforcement-learning problem, i.e., what are the states, actions, and goal, and the tuning of learning parameters, such as discretization, policy followed, and discount factor, which is the focus of this paper for the morphing vehicle problem.

Discretizing the state and action spaces is a common way to cast a continuous state and action space problem as a reinforcement-learning problem. A simple learning problem can be easily discretized into a relatively small number of states. The learned value or action-value function is generally a good representation of the agent's knowledge of the environment. A problem becomes more complex as the number of state variables needed to represent the environment increases. The number of states in the action-value function depends on how a problem is discretized. There is a trade

off, however. If the agent can only store knowledge in a small number of states, important details of the environment may be lost. If the agent can store knowledge in a very large number of states, details of the environment are captured quite well. The caveat is that the rate of convergence drops drastically as the number of states increases. Examples of state-space discretization are given in [1], which describes a space robot problem in which the orientation and the action set of the spacecraft has been discretized to facilitate learning [2], which describes quad-Q-learning in which a state-space is discretized and then sampled in a "divide and conquer" technique. Many other such examples exist, but few present a detailed analysis of the convergence properties of levels of discretization on the particular problem in question.

The policy followed by the agent is also an important aspect of any learning problem. Many unique solutions exist to handle the exploration/exploitation policy dilemma of both discrete and continuous reinforcement-learning problems. One such solution is to integrate the Metropolis Criterion into Q-learning, which eliminates some exploration blindness [3,4]. A greedy exploration policy based on the state balance criterion can also be used [5]. Exploration and exploitation can also be decoupled to cope up with any instabilities in the Q-learning algorithm [6]. An exploration algorithm that considers "prediction accuracy requirements" during exploration has been applied to a robot juggling Q-learning problem [7]. A Q-learning agent can also be restricted to explore only those options that are likely to avoid any unnecessary risk, which allows the algorithm to balance competing objectives and find satisficing solutions [8]. Many of these methods show good convergence properties and could be considered as candidate policies.

Reinforcement learning has also been applied to the problem of air vehicle morphing for mission adaptation. Valasek et al. [9] describe a methodology that combines structured adaptive model inversion with reinforcement learning to address the optimal shape change of an entire vehicle. The method learns the commands for two independent morphing parameters that produce the optimal shape. The authors show that the methodology is capable of learning the required shape and changing into it and accurately tracking some reference trajectory [9]. This methodology is further developed in [10]. It is extended to an "air vehicle" using Q-learning to learn the optimal shape change policy. The authors show that the methodology is able to handle a hypothetical 3-D smart aircraft that has two independent morphing parameters, tracking a specified trajectory, and autonomously morphing over a set of shapes corresponding to flight conditions along the trajectory [10]. Finally, the methodology is further improved upon by applying sequential function approximation to generalize the learning from previously experienced quantized states and actions to the continuous state-action space [11]. The authors showed that the approximation scheme resulted in marked improvements in the learning as opposed to the previously employed K-nearest neighbor approach. All of these examples, however, have only two independent degrees-of-freedom that must be learned and are in fact treated as separate learning problems each with only one degree-of-freedom to be learned. Learning to manipulate more morphing parameters that are interdependent and cannot be separated into independent problems creates a more complex problem. The increased dimensionality and inherent convergence issues become a concern.

Biologically inspired morphing is of great interest in the realm of micro air vehicles (MAVs). Because of their small size and membrane-lifting surfaces, they often do not have conventional control surfaces, especially on the wing. Thus, other means of control must be investigated. Some learning methods have been applied to optimize control techniques for MAVs. Flight tests show that wing twist and/or curl provide an excellent strategy to command roll maneuvers [12,13]. The torque rods used to achieve wing twist are optimized in [14], using genetic algorithms in which a vortex lattice method is used to determine fitness. However, the genetic algorithms are used to determine the placement of the torque rods to achieve the most efficient morphing of the wing rather than *how* to morph the wing to achieve the desired effects. Traditional control and optimization techniques and human pilots are still necessary to control the shape change of the aircraft.

The problem of a morphing airfoil was investigated by Hubbard [15]. The focus is on the physical shape change of an airfoil modeled by a space/time transform parameterization. The space/time parameterization results in a spatially decoupled system with Fourier coefficients as inputs and orthogonal basis shapes as outputs [15]. This is a novel concept for the actuation of an airfoil's shape. However, it requires that the final desired shape be specified and the design of controllers to send commands to the actuators.

This paper extends and analyzes the basic architecture developed in [16] in which the morphing airfoil is first cast as a simple reinforcement-learning problem and various aerodynamic goals are learned. Considering a computational model of an airfoil that calculates the aerodynamic and structural properties as the airfoil changes shape, reinforcement learning is used to learn the commands that produce the optimal shape based on lift, drag, and moment, all of which

are dependent on flight condition. Every learning problem is different, and the method applied must be tailored to that problem. The discretization of the state-space must be tuned such that good convergence is achieved while still capturing the important features of the state-space. The policy and learning parameters must also be chosen such that the action-value function converges to a near optimal solution.

The contribution of this paper is to further examine the learning problem of the morphing airfoil in detail by changing various learning parameters, such as the discretization, policy, and discount rate, while keeping the goal the agent is to learn constant. Casting the morphing problem as a reinforcement-learning problem allows a greater versatility than other methods. The shape needed to meet some performance standard for a flight phase and the path the shape parameters are to follow from some initial shape to that final shape is learned together. Previous morphing vehicle work involves designing a set of configurations for several flight phases and then designing a controller to transition from one shape to another. Problems can arise with this method if the designed shapes are not achieved and the difference is significant enough that the designed controllers no longer perform satisfactorily.

As a reinforcement-learning problem, a control policy can be learned by designing the problem such that a shape must be achieved that meets the requirements from *anywhere* in the conceivable shape state-space. In addition, the vehicle can continue to learn on-line so that as conditions change or as the vehicle parameters themselves change, the policy can learn and adapt to continue to perform well. Only the goal, constraints, and learning parameters must be set to produce such a control policy. This is especially helpful for MAVs in which design and control is a challenge due to such issues as unsteady aerodynamics. The learned control policy will inherently account for that as it is learned. The analysis described in this paper is one more step toward this overall goal. The tuning of learning parameters is necessary to achieve convergence to a usable policy. It is necessary to understand the effects of these parameters for a small, though still complex morphing problem, before tackling a morphing air vehicle problem with a greater number of shape-changing parameters. When this methodology is fully developed, it can act as a wrapper for simulated or hardware vehicles, such as a morphing MAV or Hubbard's airfoil. Through reinforcement learning, the vehicle will learn what shapes it needs to achieve to meet some goal and the control policy that will give commands to the actuators to transition from any shape to the desired shape. This analysis identifies which arrangement is most conducive to good convergence properties for the morphing airfoil reinforcement-learning problem.

This paper is organized as follows. Section II describes the mechanics of reinforcement learning and how it is implemented in Q-learning in particular. Reinforcement learning learns the optimality relations between the aerodynamic requirements and the shape. The airfoil can then be subjected to a series of aerodynamics requirements and use the relations learned to choose a good shape for the current set of requirements. The method used to discretize a continuous learning domain is developed. Section III develops the airfoil model used by the reinforcement-learning agent. This section describes the methodology used to calculate the aerodynamic and structural properties of the airfoil. Flexibility is allowed in the sense that thickness, camber, location of maximum camber, and angle-of-attack all have the potential of being commanded and used to determine the optimal configuration. Section IV describes how the airfoil model and the reinforcement-learning agent are tied together to form a morphing airfoil. Section V describes how the testing simulations are conducted once learning is completed. Section VI takes the fully developed morphing airfoil and interprets numerical examples generated from it. The numerical examples show the airfoil autonomously morphing into optimal shapes corresponding to specified aerodynamic requirements. Finally, conclusions are drawn from the numerical examples in Sec. VII.

II. Reinforcement Learning

Reinforcement learning is learning through interaction with the environment to achieve a goal. More specifically it is learning to map situations to actions to maximize some numerical reward. The learner or decision-maker is the *agent* and does not know what actions to take *a priori* as is common in most forms of machine learning. Everything outside of the agent comprises the *environment*. The agent's task is to learn a policy or control strategy for choosing actions that achieves its goals. To learn the correct policy, which is a state to action mapping, $\pi : S \rightarrow A$, the agent receives a *reward*, or reinforcement, from the environment [17].

The agent and environment interact continually is a series of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, a series of events occur. The agent first receives some representation of the environment's state, $s_t \in S$, where S is the set of all possible states. Based on the state, the agent chooses an action, $a_t \in A(s)$, where $A(s)$ is the set of

actions available to the agent in state s_t. At the next time step, the agent receives a numerical reward, $r_{t+1} \in \Re$, and is in a new state, s_{t+1}.

As the agent moves from state-to-state selecting actions and receiving rewards, it generates a mapping, as stated earlier, of states to probabilities of selecting each possible action. This policy, $\pi_t(s, a)$, is the probability that $a_t = a$ at $s_t = s$ [17].

The agent seeks to maximize the reward it receives, or more formally, its expected return, R_t. The simplest form of R_t is the sum of the rewards received after time t as shown in the following equation:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \tag{1}$$

where T is the final time step, assuming there is a final step. This breakdown of a sequence into a finite number of steps is called an *episode*. Discounted return denotes the sum of discounted rewards the agent tries to maximize, Eq. (2)

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

where γ is the discount rate and $0 \leqslant \gamma \leqslant 1$. The discount rate effectively modulates the importance of future expected rewards. If $\gamma = 0$, the agent seeks only to maximize immediate rewards. As γ approaches 1, the agent takes future rewards into account more strongly [17].

In this research, and for many reinforcement learning problems, it is assumed that the problems can be modeled as Markov decision processes (MDPs) and cast in the reinforcment-learning problem framework. An MDP satisfies the following conditional probability distribution function:

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0\} = \Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{3}$$

for all $s_1, \ldots, s + t + 1$ and for all integers $t > 0$ [18]. This means that rather than the transition to state s' and receiving reward r depending on all past states, actions, and rewards, the transition to state s' and receiving reward r is *only* dependent on s_t and a_t. A problem is considered an MDP if all the information necessary for the agent to make a decision is incorporated in the current state. The decision is not based on any past states visited, and is therefore path independent.

An underlying theme of almost all algorithms used to solve reinforcement-learning problems is estimating *value functions*. A value function is a function of the state or of state-action pairs that estimates how good it is, in terms of future rewards, for the agent to be in a given state [17]. The value of a state s under some policy π is denoted $V^\pi(s)$ and is the expected return of starting in s and following π for all subsequent steps. This expectation is formalized in the following equation:

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \tag{4}$$

A similar relationship exists for action-value functions, $Q^\pi(s, a)$, which is defined as the value of taking action a in state s under policy π. This relationship is shown in the following equation:

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\} \tag{5}$$

These can be estimated from experience and usually satisfies some recursive relationship. This relationship for the value function is derived as given below and holds for any policy π and any state s.

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \end{aligned}$$

$$= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right\}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right\} \right]$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \tag{6}$$

where $\mathcal{P}^a_{ss'}$ is the probability of transition from state s to state s' under action a, and $\mathcal{R}^a_{ss'}$ is the expected immediate reward on transition from s to s' under action a. Equation (6) is referred to as the Bellman equation for V^π. The Bellman equation for Q^π is

$$Q^\pi(s,a) = E_\pi \{ R_t | s_t = s, a_t = a \}$$

$$= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\}$$

$$= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \right\}$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_a \pi(s',a') E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s', a_{t+1} = a' \right\} \right]$$

$$= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_a \pi(s',a') Q^\pi(s',a') \right] \tag{7}$$

The next logical step is to define the optimal value function and optimal action-value function. This starts with the assumption that there is an optimal policy, π^*, better than all the others. One policy is better than another if its expected return is greater [17]. The optimal value function, V^*, is thus defined as

$$V^*(s) = \max_\pi V^\pi(s) \tag{8}$$

for all $s \in S$. Similarly, the optimal action-value function is defined as

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{9}$$

for all $s \in S$ and $a \in A(s)$. Here Q^* can be written in terms of V^* because Eq. (9) is the expected return of taking action a in state s and following an optimal policy for all subsequent steps

$$Q^*(s,a) = E_\pi \{ r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a \} \tag{10}$$

The related Bellman optimality equations are listed as follows:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{11}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a') \right] \tag{12}$$

Ideally one would simply solve the set of linear Bellman optimality equations to acquire an optimal value function or optimal action-value function. However, that requires that the transition probabilities, $\mathcal{P}^a_{ss'}$, and expected immediate rewards, $\mathcal{R}^a_{ss'}$, be known [17]. That is often not the case, unfortunately, so other methods are often employed.

There are a number of ways to solve for the value or action-value functions. Three basic solution methods are dynamic programming (DP), Monte Carlo methods, and temporal-difference (TD) learning [17].

Dynamic programming refers to algorithms that computes optimal policies given a perfect model of the environment. These are often computationally expensive and depend on a *perfect* model. Dynamic programming algorithms include policy evaluation, policy improvement, policy iteration, value iteration, etc. [17].

Monte Carlo methods estimate the value functions and try to find optimal policies. One advantage of these methods over basic DP is that they require only experience and not perfect knowledge of the environment. Learning can be conducted on-line or in simulation with no prior knowledge of environment dynamics. These methods learn based on an episode by episode averaging of sample returns. Monte Carlo methods include Monte Carlo policy evaluation, Monte Carlo estimation of action values, Monte Carlo control (both on-policy and off-policy), etc. [17].

Temporal-difference learning can be thought of as a combination of ideas from both DP and Monte Carlo [17]. These methods learn from raw experience without the need for a model of the environment's dynamics. They bootstrap in the sense that estimates are updated *during* an episode based on other learned estimates. Temporal-difference learning methods include TD prediction, Sarsa, Q-learning, actor-critic methods, etc.

A. Implementation of Reinforcement-learning Agent

For the present research, the agent in the morphing airfoil problem is its RL agent. It attempts to independently maneuver from some initial state to a final goal state characterized by the aerodynamic properties of the airfoil. To reach this goal, it endeavors to learn, from its interaction with the environment, the optimal policy that, given the specific aerodynamic requirements, commands the series of actions that changes the morphing airfoil's thickness or camber toward an optimal one. The environment is the resulting aerodynamics the airfoil is subjected to. It is assumed that the RL agent has no prior knowledge of the relationship between actions and the thickness and camber of the morphing airfoil. However, the RL agent does know all possible actions that can be applied. It has accurate, real-time information of the morphing airfoil shape, the present aerodynamics, and the current reward provided by the environment.

The RL agent uses a one-step Q-learning method, which is a common off-policy TD control algorithm. In its simplest form, it is a modified version of the following equation and is defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{13}$$

The Q-learning algorithm is illustrated as follows [17]:

Q-Learning()

- Initialize $Q(s, a)$ arbitrarily
- Repeat (for each episode)
  - Initialize s
  - Repeat (for each step of the episode)
    - Choose a from s using policy derived from $Q(s, a)$ (e.g. ε-Greedy Policy)
    - Take action a, observe r, s'
    - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$
    - $s \leftarrow s'$
  - until s is terminal
- return $Q(s, a)$

The agent learns the greedy policy, defined as:

$$\begin{aligned} &\varepsilon\text{-greedy policy} \\ &\text{if(probability} > 1 - \varepsilon) \\ &a = \arg\max_a Q(s, a) \\ &\text{else} \\ &a = rand(a_i) \end{aligned} \tag{14}$$

As the learning episodes increase, the learned action-value function $Q(s, a)$ converges asymptotically to the optimal action-value function $Q^*(s, a)$. The method is an off-policy one as it evaluates the target policy (the greedy policy) while following another policy. The policy used in updating $Q(s, a)$ can be a random policy, with each action having the same probability of being selected. The other option is an ϵ-greedy policy, where ϵ is a small value. The action a with the maximum $Q(s, a)$ is selected with probability $1 - \epsilon$, otherwise a random action is selected.

If the number of the states and the actions of an RL problem is a small value, its $Q(s, a)$ can be represented using a table, where the action-value for each state-action pair is stored in one entity of the table. Because the RL problem for the morphing vehicle has states (the shape of the airfoil) on continuous domains, it is impossible to enumerate the action-value for each state-action pair. In essence, there are an infinite number of state-action pairs. One commonly used solution is to artificially quantize the states into discrete sets, thereby reducing the number of state-action pairs the agent must visit and learn. The goal in doing this is to reduce the number of state-action pairs while maintaining the integrity of the learned action-value function. For a given problem, experimentation must be conducted to determine what kind of quantization is appropriate for the states.

In this paper, several increasingly larger quantizations are considered to determine what the effect of discretization. For the problem at hand, the state of the morphing airfoil cast as a reinforcement-learning problem are the shape parameters—thickness and camber. Studying the effects of the discretization of these state variables is an important stepping stone for the understanding of the problem before more state variables (i.e., shape parameters) are added in future development. The details of how the morphing airfoil problem is cast as a reinforcement-learning problem are described in Sec. IV.

B. Learning on a 2- and *N*-Dimensional Continuous Domain

Q-learning on a continuous domain quickly becomes intractable when one considers that convergence of the algorithm to the optimal action-value function is only guaranteed if the agent visits every possible state an infinite number of times [19]. An agent would therefore visit an infinite number of states using an infinite number of actions an infinite number of times. Add in the fact that the states can be defined by anywhere from 1 to N continuous variables and the dimensionality of the problem becomes a significant problem.

One way to cope up with the inherent complexity of a continuous domain-learning problem is to discretize the state-space by overlaying a pseudo-grid. The essential ideas of this concept can be best introduced in terms of a 1-dimensional problem. The notation can then be generalized for the 2- and N-dimensional problems.

For the 1-dimensional problem, the state-space can be represented by a line as seen in Fig. 1. An arbitrary set of vertices $\{{}^1X, {}^2X, \ldots, {}^kX, \ldots\}$ are introduced at a uniform distance h apart. Ideally, h is chosen such that a vertex lies on both end points of the state-space. In the learning algorithm, the agent is allowed only to visit the overlaying vertices and their corresponding states. This technique effectively reduces the state-space from infinity to a finite number of states, thus rendering the problem more manageable.

To further simplify the problem, we restrict what actions the agent may take. When the agent is at the Ith vertex $X = {}^IX$, it may only move to ${}^{I-1}X$ or ${}^{I+1}X$. Now the problem has only two possible actions rather than an infinite number, which further reduced the problem complexity.

Let L denote the length of the continuous domain. As per our formulation, there are

$$N_{V_1} = \frac{L}{h} + 1 \tag{15}$$

vertices, where N_V is the number of vertices, and two actions. Therefore, there are only

$$N_1 = 2\left(\frac{L}{h} + 1\right) \tag{16}$$

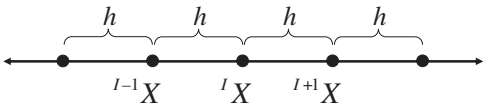


Fig. 1 1-Dimensional state-space with overlaying pseudogrid.

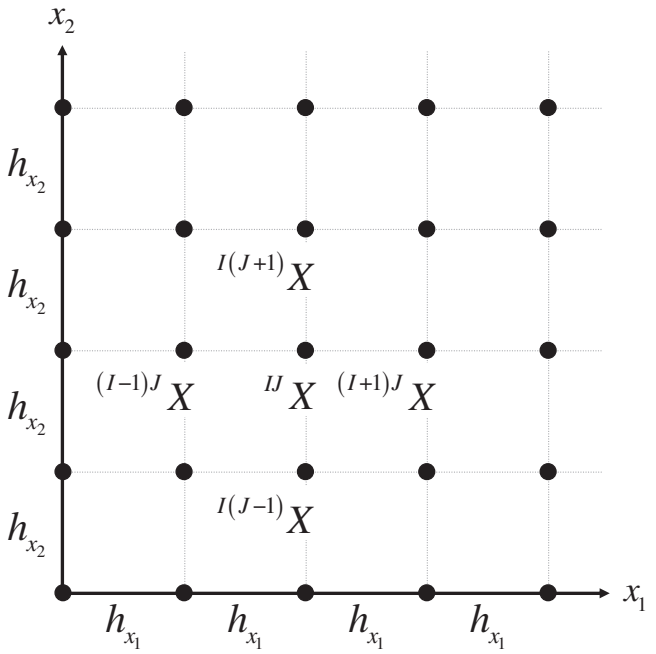


Fig. 2 2-Dimensional state-space with overlaying pseudogrid.

state-action pairs, where N is the number of state-action pairs.

The 2-dimensional problem can be represented in a similar manner. In this case, the state-space is represented by Fig. 2. An arbitrary set of vertices $\{^{11}X, ^{12}X, \ldots, ^{ij}X, \ldots\}$ are again introduced at uniform distances h_{x_1} or h_{x_2} apart. The actions available to the agent are again restricted as in the 1-dimensional case. For the 2-dimensional case, when the agent is at the IJth vertex $X = {}^{IJ}X$, it may only move to vertices $^{(I-1)J}X$, $^{(I+1)J}X$, $^{I(J-1)}X$, and $^{I(J+1)}X$, a total of 4, or $2 * 2$, actions.

This problem is more complex than the previous one, yet it is still simpler than a 2-dimensional continuous state-space problem. For this 2-dimensional discrete case, let L_{x_1} and L_{x_2} denote the length in the x_1- and x_2-direction, respectively, of the continuous domain. This results in

$$\begin{aligned} N_{V_2} &= \left(\frac{L_{x_1}}{h_{x_1}} + 1\right)\left(\frac{L_{x_2}}{h_{x_2}} + 1\right) \\ &= \prod_{i=1}^{2}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \end{aligned} \tag{17}$$

vertices. Therefore, there are

$$N_2 = 2 * 2\prod_{i=1}^{2}\left(\frac{L_{x_i}}{h_{x_i}} + 1\right) \tag{18}$$

state-action pairs. This 2-dimensional development is what will be used in the rest of this paper.

From here the formulation can be generalized to the N-dimensional case. For an N-dimensional continuous state-space, an arbitrary set of vertices $\{^{11\ldots1}X, ^{11\ldots2}X, \ldots, ^{NN\ldots N}X\}$ are introduced at uniform distances $h_{x_1}, h_{x_2}, \ldots, h_{x_N}$ apart. The actions are restricted to the two nearest vertices in any direction from the current vertex $X = {}^{IJ\cdots}X$, yielding a total of $2N$ actions available to the agent from any given vertex.

Now let $L_{x_1}, L_{x_2}, \ldots, L_{x_N}$ denote the length in the x_1-, x_2-, ..., and x_N-directions, respectively. As a result, there are

$$N_{V_N} = \prod_{i=1}^{N} \left(\frac{L_{x_i}}{h_{x_i}} + 1 \right) \tag{19}$$

vertices. Therefore, there are

$$N_N = 2N \prod_{i=1}^{N} \left(\frac{L_{x_i}}{h_{x_i}} + 1 \right) \tag{20}$$

state-action pairs.

Discretizing the domain in this way can greatly simplify a learning problem. Intuitively, the larger h_{x_i} is, the fewer the number of vertices, resulting in fewer visits by the agent necessary to learn the policy correctly. Special care must be taken, however, in the choice of h_{x_i} and the definition of the goal the agent attempts to attain. If the only goal state lies between vertices, then the agent will be unable to learn the actions necessary to reach the goal state.

The "Curse of Dimensionality" can still become a problem when using this technique. As N increases, the number of state-action pairs increases quickly. Manipulation of h_{x_i} can alleviate some problems, but can eventually become overwhelmed. However, the number of state-action pairs remains finite. In this paper, a 2-dimensional problem is analyzed. Careful manipulation of h_{x_i} helps to maintain a manageable learning problem.

C. Exploration/Exploitation Dilemma

A policy, π is the mapping of states to actions, $\pi : S \rightarrow A$, which means that the agent selects its next action a_t based on the current state s_t, or $\pi(s_t) = a_t$. When selecting the next action, one typical problem the agent has to face is the exploration–exploitation dilemma [17]. If the agent selects a greedy action that has the highest value, then it is exploiting its knowledge obtained so far about the values of the actions. If instead it selects one of the nongreedy actions, then it is exploring to improve its estimate of the nongreedy actions' values. Exploiting knowledge from the outset usually results in the agent finding and preferring local optima rather than the global goal [17]. Exploring from the outset and continuing throughout the learning process, however, avoids this problem, though the agent is more likely to continue to randomly explore areas that are not of interest [17]. Given that Q-learning is an off-policy reinforcement method, the policy followed while learning must be chosen to balance these concerns, promote efficient convergence, and tailored for the application of interest. Possible policies include greedy, nongreedy, ε-greedy (Eq. (14)), and softmax action selections.

Many unique solutions exist to handle the exploration–exploitation policy dilemma of both discrete and continuous reinforcement-learning problems. One such solution is to integrate the Metropolis Criterion into Q-learning, which eliminates some exploration blindness [3,4]. A greedy exploration policy based on the state balance criterion can also be used [5]. Exploration and exploitation can also be decoupled to cope with any instabilities in the Q-learning algorithm [6]. An exploration algorithm that considers "prediction accuracy requirements" during exploration has been applied to a robot juggling Q-learning problem [7]. A Q-learning agent can also be restricted to explore only those options that are likely to avoid any unnecessary risk, which allows the algorithm to balance competing objectives and find satisficing solutions [8]. Similar to Goodrich's satisficing Q-learning is Park and Kim's two mode Q-learning, which also separates success and failure experiences to learn more quickly. Simsek and Barto [20] propose a method efficient exploration aided by a principled heuristic. Singh et al. [21] describes in detail the proving of convergence for single-step on-policy reinforcement learning algorithms for control with both decaying exploration and persistent exploration.

For part of the analysis in this paper, the agent is allowed to randomly explore the thickness/camber state-space and receive reinforcement based on the aerodynamics of the airfoil during all learning episodes to gain an understanding of the nature of the problem. The rest of the analysis will entail changing the policy the agent uses to choose actions throughout the learning process. The agent will begin learning by using a fully random exploration process. As the agent conducts learning episodes, the agent begins to exploit its knowledge more often. This change in policy is caused by increasing ε from $0 \rightarrow \varepsilon \leqslant 1$. This annealing of ε allows the agent to explore early on to eliminate any local optima. During later episodes the agent exploits its knowledge to find the global goal.

D. Annealing of Discount Factor

The discount factor also influences convergence of the reinforcement-learning problem. The discount factor, γ, is related to the discount rate as defined by economists by as

$$\gamma = \frac{1}{1+r} \tag{21}$$

where r is the discount rate [22]. Though Lampton et al. [16] showed that a constant value of $\gamma = 0.7$ yielded good convergence properties, for high-dimensional problems annealing the discount factor can aide convergence [22]. Annealing the discount rate can even affect smaller problems as will be shown later in this paper. At low values of γ, the agent learns primarily the utility of the RL problem [22]. Incrementally increasing γ throughout the learning process allows the agent to learn how the primary costs of its actions accumulate [22]. As the methodology developed in the this and other papers is a stepping stone for higher dimensional, and more complex, learning problems, the effect of annealing γ from low to high values is analyzed.

III. Airfoil Model Representation

To calculate the aerodynamic properties of many different airfoils in a short period of time, or as a single airfoil changes shape, a numerical model of the airfoil is developed. A constant strength doublet panel method is used to model the aerodynamics of the airfoil. The main assumption is that the flow is incompressible, otherwise a much more complex model is necessary. This assumption is valid because current interests lie in the realm of MAVs, which fly at speeds less than Mach 0.3. Other assumptions are that both the upper and lower surfaces of the airfoil are pinned at the leading and trailing edge rendering the structural moment at these points to be zero. These boundary conditions become important in later sections in the calculation of M_y and σ_{xx}. One final assumption is that the flow is inviscid. Thus, the model is only valid for the linear range of angle-of-attack.

The flexibility of this type of model allows the reinforcement-learning algorithm developed to manipulate four possible degrees-of-freedom. The degrees-of-freedom are

1) Airfoil thickness
2) Camber
3) Location of maximum camber
4) Airfoil angle-of-attack

Despite this versatility, there are some limitations to the model. As the model uses a panel method to calculate the aerodynamics, it is very sensitive to the grid, or location of the panels, and the number of panels created. The grid must be a sinusoidal spaced grid in the x direction, which puts more points at the trailing edge of the airfoil. This type of grid is necessary because many aerodynamic changes occur near the trailing edge. If the number of panels were to decrease, the accuracy of the model would also decrease. However, as the number of panels generated is increased, the computational time of the model increases as well. Thus, a balance is needed between accuracy and computational time. This balance can be achieved by finding a set number of panels for which any increase from that number of panels yields a minimal accuracy increase. For example, assume 50 panels are chosen initially. If the number of panels were changed to 100 and the accuracy of the model increased by 10 percent, this increase in the number of panels would be deemed necessary. If the number of panels were changed from 100 to 150 and the accuracy of the model increased by <1 percent, then this increase in the number of panels would be deemed unnecessary, and 100 panels is chosen as the correct number of panels to use.

A full description of the airfoil model development can be found in [16]. Validation and verification of the airfoil model can also be found in [16].

IV. Morphing Airfoil Cast as a Reinforcement Learning Problem

The morphing of the airfoil entails changing the thickness and camber at this stage in the development of this methodology (see Fig. 3). The reinforcement-learning agent specifies these two parameters corresponding to the current flight condition (a.k.a. aerodynamic goal). The CFD model itself constitutes the *environment* with which the agent interacts. Thus, the state variables are the thickness and camber inputs to the CFD model.

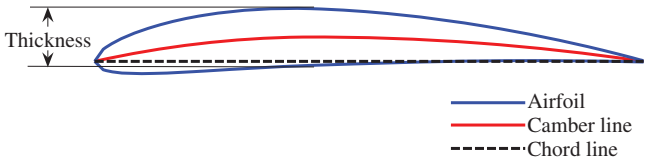


Fig. 3 Representative Airfoil.

The agent interacts with its environment by choosing actions from a set of admissible actions. The state-space is discretized in the manner described in Sec. B, so these actions include incremental changes in the shape parameters of the airfoil. Thus, the agent is effectively restricted to movement between adjacent vertices. An example of admissible action in this context is the following. The agent chooses to move in the x_1-direction from vertex ${}^{IJ}X$ in the 2-dimensional problem. For the initial discretization, the two possible actions in the x_1-direction are defined as follows:

$$\begin{aligned} A^1_{11} &\equiv {}^{(I+1)J}X - {}^{IJ}X = h^1_{x_1} \\ A^1_{12} &\equiv {}^{(I-1)J}X - {}^{IJ}X = -h^1_{x_1} \end{aligned} \tag{22}$$

Equation (22) can be summarized by saying the initial admissible actions in the x_1-direction are $A_1 = \pm h^1_{x_1}$. Similar relationships can be found for the x_2-direction. Admissible actions in the other direction is $A_2 = \pm h^1_{x_2}$. The definitions of the x_i axes for all of the examples are defined in Table 1. To read these tables consider the x_1-direction, for example. The agent changes $\pm 0.50\%$ of the chord in thickness in this direction when $h_{x_1} = 0.50\%$.

The *goal*, g, of the agent for this problem is defined by the aerodynamics of the airfoil. Every goal has a range, g_r, associated with it. The numerical example to follow has goals defined by the airfoil lift coefficient, c_l.

The reward structure for this problem is a traditional negative, neutral, and positive reinforcement scheme and was used in the early stages of this research. This scheme is summarized in Table 2. The $s < \text{limits}_{\text{min}}$ or $s > \text{limits}_{\text{max}}$ refers to when the state, s is beyond the bounds or limits of the of the state-space. These limits are listed in Table 3.

Thus, the morphing airfoil problem is cast as a reinforcement-learning problem. The agent chooses and action that changes the state (thickness and camber) of the airfoil. The aerodynamics of the airfoil are calculated by the environment, and the agent "senses" the lift, drag, and/pitching moment. The appropriate reinforcement is calculated based on the current aerodynamics, the goal aerodynamics, and any constraints. Finally, the update equation, Eq. (13), is used to update the knowledge base learned by the agent. Then the process repeats. The agent will learn not only

Table 1 Morphing airfoil axis definitions

x_i	Definition
x_1	Thickness (%)
x_2	Camber (%)

Table 2 Morphing airfoil reward structure

Bounds	Reward
$g - g_r \leqslant c \leqslant g + g_r$	20
$s < \text{limits}_{\text{min}}$ or $s > \text{limits}_{\text{max}}$	−20
Otherwise	0

Table 3 Morphing airfoil parameter limits

Limit	Lower	Upper
Thickness (% chord)	10	18
Camber (% chord)	0	5

shape(s) that meet the aerodynamic goal specified by the user, it will also learn the path or control policy to maneuver from any thickness/camber pair within the state-space to a shape that meets the goal.

V. Monte Carlo Simulation

The action-value function is analyzed by conducting a performance-based policy comparison using a set of simulations using the learned function. The performance-based policy comparison measures the performance by conducting a set of Monte Carlo simulations. It is referred to as Monte Carlo in the sense that initial conditions are taken from a uniform distribution and a large number of simulations are conducted and recorded. In each simulation, the agent is initialized in a random nongoal state within the region of the current level of discretization. It then uses the current learned greedy policy, meaning that it exploits its current knowledge of the state-space, to navigate through the state-space to find the goal. A *success* occurs when the agent navigates from the random initial state to a goal state without encountering a boundary. A *failure* occurs when the agent either encounters the outer most boundary of the state-space, the boundary of the current level of discretization, or gets "lost" and wanders around the state-space. This simulation is conducted a predefined number of times, usually 500 simulations in this research, and each success is recorded. The success percentage is then calculated as follows:

$$\%\ \text{Success} = \frac{\#\ \text{of successes}}{\text{Total}\ \#\ \text{of simulations}} \tag{23}$$

When this success percentage is above some threshold, usually 98% in this research, the second stopping criterion is satisfied. Both the first and the second stopping criterion must be met for the learning at the current level of discretization to be terminated and learning continued at a finer level of discretization as necessary.

VI. Numerical Examples

The purpose of the numerical examples is to demonstrate the learning performance of the reinforcement-learning agent when integrated with the aerodynamic model. The agent follows either a 100% exploration policy, an ε-greedy policy with annealing, or an ε-greedy annealing policy in addition to annealing of the discount factor. The agent is to learn the respective action-value functions using each one of the four sets of h_{x_i}. For the purpose of direct comparison between the discretized state-space effected by the different h_{x_i} sets, the chord, angle-of-attack, number of episodes, boundary reward, goal reward, and aerodynamic goal are the same for each round of learning. The values of these constants are listed in Table 4.

For each of the 5000 episodes, the agent begins in a random initial state that is not classified as a goal state. It explores the state-space of thickness–camber combinations until it hits the predefined limit of total number of actions or finds a goal state. Should the agent run into a boundary, that boundary location is noted, and the agent chooses another action. The actions for all four cases are defined by the h_{xi}, listed in Table 5.

The learning performance is analyzed in several ways. First, the dimensionality of the action-value functions are compared between the action sets. Next, the simulation results for different policies are compared for each set of h_{x_i} to examine both the learning performance for a particular set of h_{x_i}. The simulation results are then recast such that comparisons between sets of h_{x_i} for each policy can be more easily made to again examine the learning performance. Finally, the final value functions are compared and analyzed.

Table 4 Parameter constants

Parameter	Value
Chord	1 m
Angle-of-attack	2.0°
Episodes	5000
g	$c_l \geqslant 0.4$
α	0.01
γ	0.7

Table 5 Distance between adjacent vertices

Cases	h_{x_1}	h_{x_2}
Case 1	0.10	0.10
Case 2	0.25	0.25
Case 3	0.50	0.50
Case 4	1.00	1.00

Table 6 Nongoal states and state-action pairs

h_{x_i}	States	State-action pairs
0.10	3379	13,516
0.25	530	2120
0.50	153	612
1.00	45	180

A. Dimensionality

The dimensionality of the problem to be learned is an ever present concern for learning algorithms. A problem with a high number of states, or in the case of Q-learning, a high number of state-action pairs, necessitates a greater number of learning episodes, and thus more computational time, to learn the required action-value function. Conversely, fewer states mean faster learning, especially given that computational aerodynamic models are usually computationally

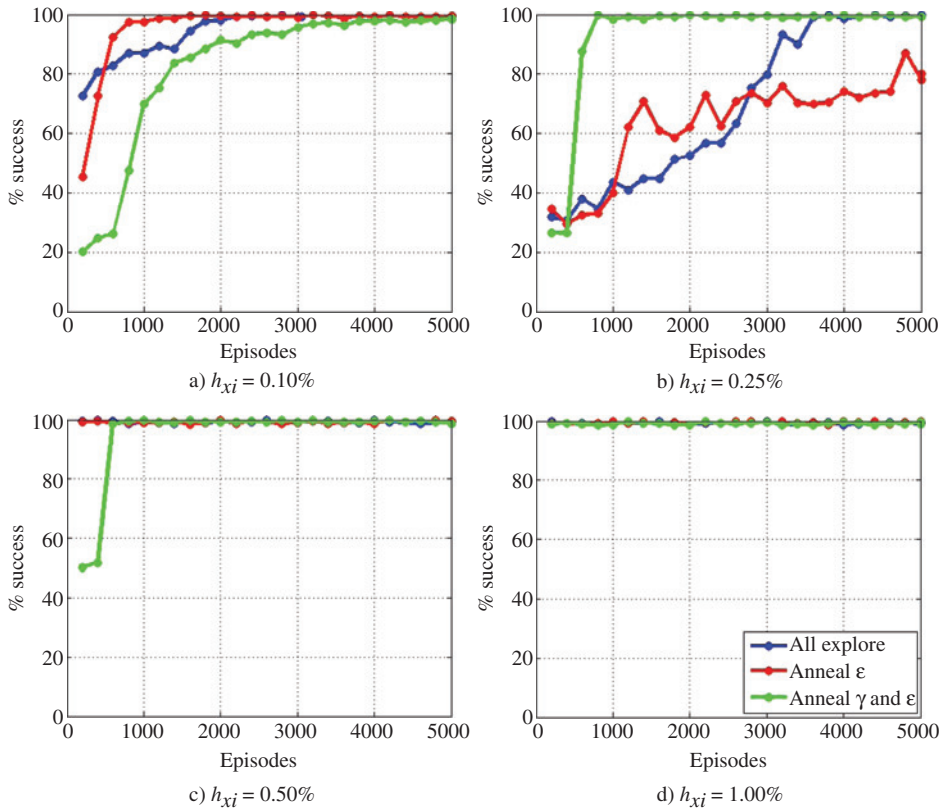


Fig. 4 Policy comparison for each h_{x_i}.

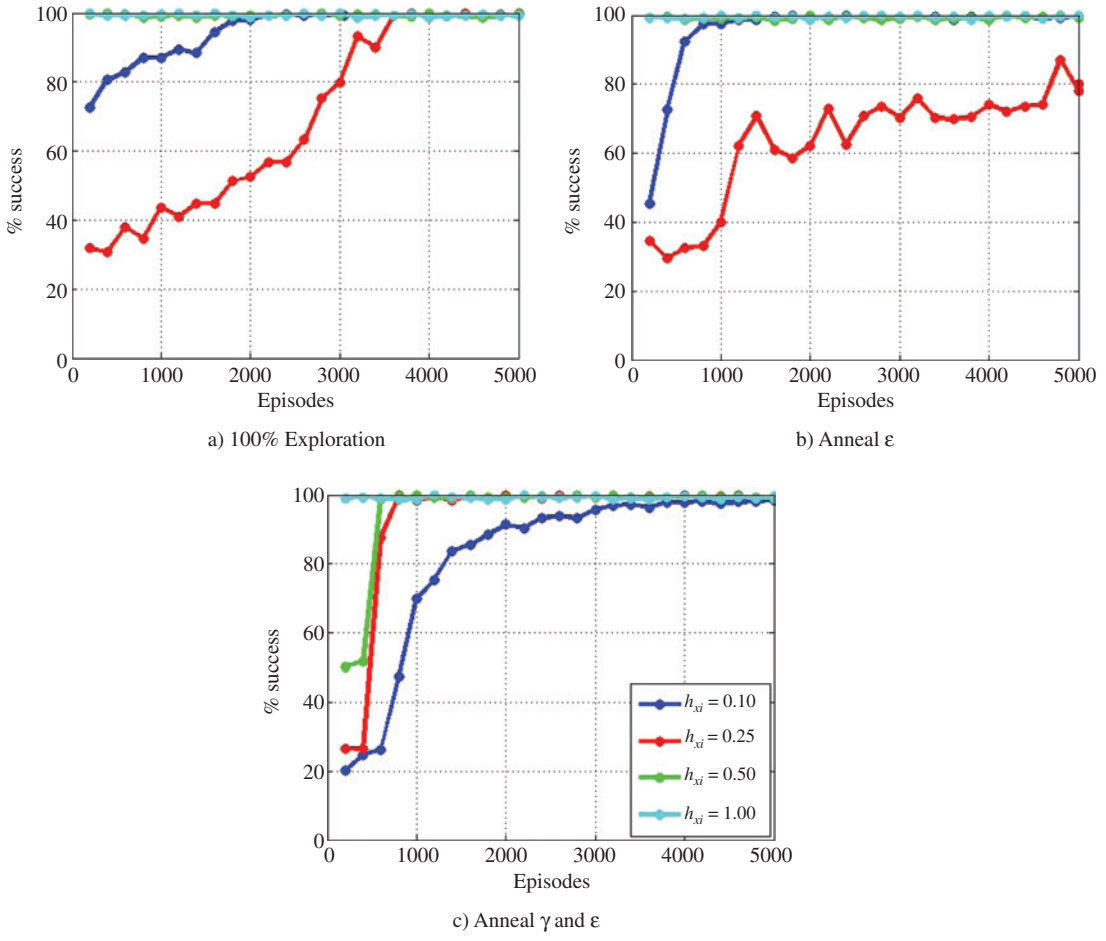
intensive. The caveat is that there must be enough states to fully capture the details of the action-value function for the problem at hand. The number of nongoal states and state-action pairs for each value of h_{x_i} listed in Table 5 are shown in Table 6.

B. Policy Comparison and Analysis

This section considers a direct comparison for each h_{x_i} between action-value functions when the agent follows one of the following:

1) 100% exploration policy,
2) ε-greedy policy with annealing,
3) ε-greedy annealing policy and annealing of the discount factor, γ.

The results of the simulations described in Sec. V are shown in Figure 4. All four subfigures show that the action-value function in which the agent followed an annealing policy converges to a good solution the quickest. A 96–100% success rate is achieved in as little as 200 episodes for the coarsest discretization and 800 episodes for the finest discretization. The results of the learning in which the discount factor also anneals shows a much lower success rate early in the learning, then converging quickly after a couple hundred episodes. In this case, convergence to a 95–100% success rate is achieved in 200–3000 episodes. The most distinct differences between policies are seen in Fig. 4a. Rate of convergence is expected to be lower, given the larger number of states the agent must visit. Figure 4d suggests that when the agent must visit only a small number of states, annealing the discount factor and/or the policy



a) 100% Exploration

b) Anneal ε

c) Anneal γ and ε

Fig. 5 h_{x_i} comparison.

makes little difference in the rate of convergence. The "kink" evident in Fig. 4b is most likely a result of the small probability that the agent chooses a random action and encounters a boundary.

C. h_{x_i} Comparison and Analysis

This section takes a different approach than the previous section and considers a direct comparison between the action-value functions for each h_{x_i}. The results presented in this manner are shown in Fig. 5. The figure shows that there is the most difference in convergence between h_{x_i} values when the policy is fully explorative and when both the policy and discount factor changes as learning progresses; 5000 and 3000 episodes, respectively, are required for all four action-value functions to converge to at least a 95% success rate. Generally, convergence for these cases increases as h_{x_i} increases. This result is again due to the decreasing number of states the agent must visit and update. Figure 5b shows the highest rate of convergence for all values of h_{x_i}. All four action-value functions converge to nearly a 100% success rate within 800 episodes. This fact suggests that for this problem annealing the policy from fully explorative to almost fully exploitative yields the best results.

D. Value Function Analysis

It is helpful to consider a value function approximated from the learned action-value function. The value function can show how the learning differs when learning parameters or discretization is modified. The value function is calculated using the following equation:

$$V(s) \approx \max_a Q(s, a) \tag{24}$$

The value functions calculated for each of the final action-value functions presented in the previous sections are illustrated in this section.

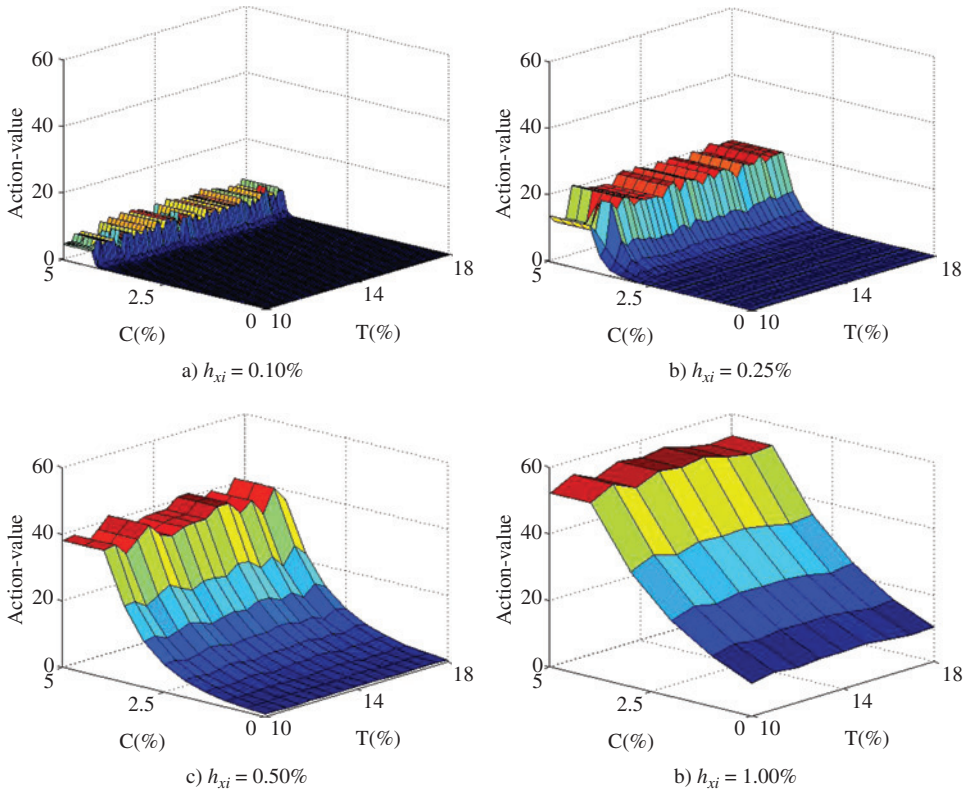


Fig. 6 Value Functions for 100% Exploration.

Figure 6 shows the value functions when the agent follows a fully explorative policy. The general shapes of each of the value functions are similar. The main difference shown is that the maximum value in the value functions range from 10 to almost 60 as h_{x_i} increases. The reason is that with fewer states to visit, the agent is able to visit and reinforce every state more often than if there are a larger number of states. Also, when there are a larger number of states, it takes more visits to each state for the rewards to propagate back into neutral or nongoal states. This results in the sharp decline in value from the area near the goal to the small values for nongoal states farther from the goal. Notice the sharpness of the decline lessens as h_{x_i} increases. The smoothness of the functions is a result of the policy allowing the agent the chance to visit each state an equal number of times. (Note: In Figs 6, 7, and 8, C is an abbreviation for Camber and T is an abbreviation for thickness.)

Figure 7 shows the value functions for a policy annealing from fully explorative to mostly exploitative. Notice the shapes of the value functions now differ between each h_{x_i}. Figure 7a shows two prominent peaks in the value function. This results from an unequal distribution of state visits early on in the learning. At the outset the agent explores randomly, but given the number of state-action pairs, the agent does not necessarily have time to evenly propagate rewards through the action-value function. For this example of learning, the agent visited the states near the peaks of Fig. 7a early. As the policy changed to require the agent to exploit its knowledge more often, the agent favored the two areas it had already explored the most. This effect resulted in the two peaks. A similar phenomenon occurred for $h_{x_i} = 0.25\%$ and is shown in Figure 7b. There are a couple small peaks that are not as extreme as those seen in Fig. 7a. This result means that the agent was able to more evenly explore all the states before the policy changed. Given the fewer number of states, this is to be expected. The trend continues as h_{x_i} increases. Figure 7d is very similar to Fig. 7d. From this figure, it is evident that as the number of states decrease, the policy the agent follows while learning affects the final action-value function less and less.

Figure 8 shows the value functions resulting from both an annealing policy as well as the discount factor, γ, increasing from 0.0 to 0.7 as learning progresses. Figure 8a shows a similar prominent peak in the value function as

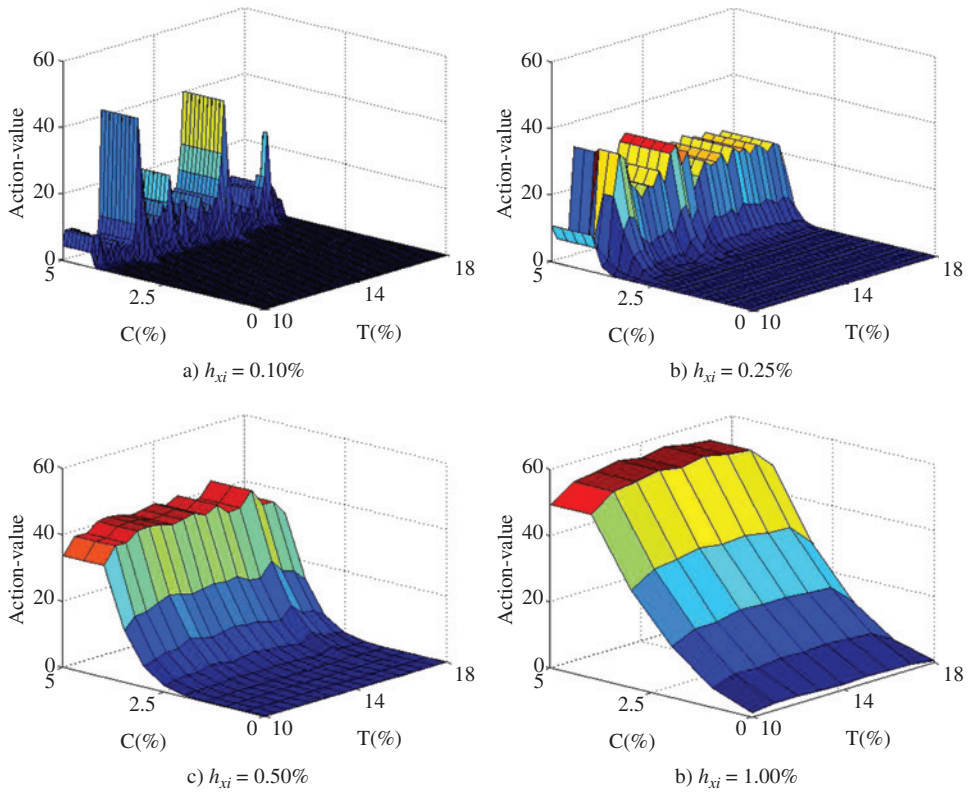


Fig. 7 Value functions for annealing of ε.

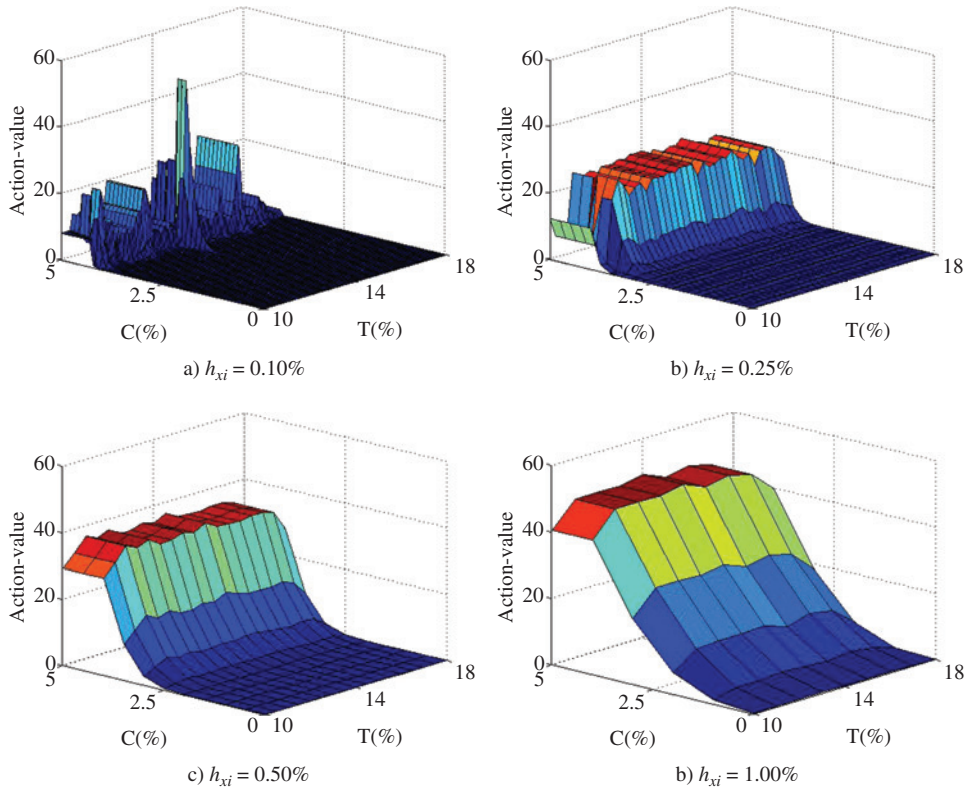


Fig. 8 Value functions for annealing of ε and γ.

that seen in Fig. 8a. This peak is again a result of the policy followed and the total number of states. A peak is not evident in Fig. 8b as it was in Fig. 8b. One reason is that the discount factor early in the learning is equal to or near zero. When the discount factor is equal to zero, the agent updates the action-value based only on whatever reward it receives. No update in the form of $\gamma \max_{a'} Q(s', a')$ is added to the value function. Another reason is the policy followed as noted previously. Once again as the number of states reduces to that shown in Fig. 8d, the value function is not appreciably different from the value functions in Figs. 6d and 7d.

VII. Conclusions

This paper analyzes a technique for learning the optimal shape change policy of a morphing airfoil that combines machine-learning and analytical aerodynamic calculations. Effects of changing learning parameters such as the policy the agent follows and the discount factor are investigated, along with the effects of the discretization of the state-space. With these changes, the agent learns the policy to morph in two interdependent degrees-of-freedom, thickness and camber, from some initial state to the defined goal state. The effects the differences in policy, discount factor, and discretization were then presented and discussed.

The results show that the agent following the policy annealing from fully *explorative* to almost fully *exploitative* yielded the fastest convergence of the action-value function, regardless of the value of the action step size. Convergence to a success rate between 96 and 100% was reached in as little as 200 epsiodes and as many as 800 episodes. This policy is a promising candidate to handle the increasing number of states as the morphing airfoil problem becomes more complex. An action step size value of 1.00% resulted in the fastest convergence rate, regardless of policy followed or discount factor. A nearly 100% success rate was achieved within 200 episodes in all instances. The agent tends to favor particular goal states when the discount factor and/or the policy is annealed and the discretization of the state-space is fine. The reinforcement-learning agent is sensitive to the discount factor, the policy followed by the

agent, and discretization of the state space. If the discretization is very fine, then rate of convergence is slower, taking upwards of 3000 episodes to achieve a 96% success rate, than when the discretization is coarse, only 200 episodes needed to achieve a 100% success rate.

The results show that the morphing airfoil problem is most sensitive to the policy followed and the discretization of the state-space, which is usually the case for reinforcement learning problems. As a consequence, these two learning parameters must be carefully tuned based on these findings for future development of the morphing airfoil/vehicle problem specifically. Though several thousand episodes is not a large amount for many reinforcement learning problems, the morphing vehicle problem is different in the sense that change the shape and calculating or sensing the aerodynamics takes time, which means conducting tens of thousands or even millions of episodes is not feasible. The results of this paper show that faster convergence can be attained with a larger discretization and some policy manipulation, though there is a loss of detail. Conversely, a high level of precision is possible with a finer discretization and more episodes. On the basis of these results and the nature of the problem, combining both discretizations in a progressive manner may meld the best of both: fewer episodes needed for convergence and a high level of precision with respect to the goal.

Acknowledgment

This work was sponsored (in part) by the Air Force Office of Scientific Research, USAF, under grant/contract number FA9550-08-1-0038. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

[1] Senda, K., Matsumoto, T., Okano, Y., Mano, S., and Fujii, S., "Autonomous Task Achievement by Space Robot Based on Q-Learning with Environment Recognition," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, 11–14 Aug. 2003, AIAA-2003-5462.

[2] Clausen, C., and Wechsler, H., "Quad-Q-Learning," *IEEE Transactions on Neural Networks*, Vol. 11, No. 2, Feb. 2000, pp. 279–294.
doi: 10.1109/72.839000

[3] Guo, M., Liu, Y., and Malec, J., "A New Q-Learning Algorithm Based on the Metropolis Criterion," *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, Vol. 34, No. 5, Oct. 2004, pp. 2140–2143.
doi: 10.1109/TSMCB.2004.832154

[4] Gao, Q., and Hong, B., "An Improved Q-learning Algorithm Based on Exploration Region Expansion Strategy," *Proceedings of the 6th World Congress on Intelligent Control and Automation*, Dalian, China, 21–23 June 2006.

[5] Zheng, Y., Luo, S., and Zhang, J., "Greedy Exploration Policy of Q-learning based on State Balance," *IEEE Region 10 Annual International Conference*, Melbourne, Australia, 21–24 Nov. 2005.

[6] Nowe, A., Steenaut, K., Fakir, M., and Verbeeck, K., "Q-learning for Adaptive, Load based Routing," *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics. Part 4*, San Diego, CA, 11–14 Oct. 1998.

[7] Schaal, S., and Atkeson, C., "Robot Juggling: Implementation of Memory-based Learning," *IEEE. Control Systems Magazine*, Vol. 14, No. 1, Feb. 1994, pp. 57–71.
doi: 10.1109/37.257895

[8] Goodrich, M. A., and Quigley, M., "Satisficing Q-learning: Efficient Learning in Problems with Dichotomous Attributes," *Proceedings of the 2004 IEEE International Conference on Machine Learning and Applications*, Louisville, KY, 16–18 Dec. 2004.

[9] Valasek, J., Tandale, M., and Rong, J., "A Reinforcement Learning — Adaptive Control Architecture for Morphing," *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 4, April 2005, pp. 174–195.
doi: 10.2514/1.11388

[10] Tandale, M., Rong, J., and Valasek, J., "Preliminary Results of Adaptive-Reinforcement Learning Control for Morphing Aircraft," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, 16–19 Aug. 2004, pp. 3215–3225, AIAA-2004-5358.

[11] Valasek, J., Doebbler, J., Tandale, M., and Meade, A., "Improved Adaptive-Reinforcement Learning Control for Morphing Unmanned Air Vehicles," *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, Vol. 38, No. 4, Aug. 2008, pp. 1014–1020.
doi: 10.1109/TSMCB.2008.922018

[12] Garcia, H. M., Abdulrahim, M., and Lind, R., "Roll Control for a Micro Air Vehicle Using Active Wing Morphing," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, 11–14 Aug. 2003, AIAA-2003-5347.

[13] Abdulrahim, M., Garcia, H. M., Ivey, G. F., and Lind, R., "Flight Testing a Micro Air Vehicle Using Morphing for Aeroservoelastic Control," *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, Palm Springs, CA, 19–22 April 2004, pp. 1776–1792, AIAA-2004-1674.

[14] Stanford, B., Abdulrahim, M., Lind, R., and Ifju, P., "Investigation of Membrane Actuation for Roll Control of a Micro Air Vehicle," *Journal of Aircraft*, Vol. 44, No. 3, May–June 2007, pp. 741–749.
doi: 10.2514/1.25356

[15] Hubbard, J. E., Jr. "Dynamic Shape Control of a Morphing Airfoil Using Spatially Distributed Transducers," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 3, 2006, pp. 612–616.
doi: 10.2514/1.15196

[16] Lampton, A., Niksch, A., and Valasek, J., "Reinforcement Learning of Morphing Airfoils with Aerodynamic and Structural Effects," *Journal of Aerospace Computing, Information, and Communication*, Vol. 6, No. 1, Jan. 2009, pp. 30–50.

[17] Sutton, R., and Barto, A., *Reinforcement Learning — An Introduction*, The MIT Press, Cambridge, MA, 1998, pp. 9, 51–80, 89–100, 111, 133, 148–150.

[18] Stark, H., and Woods, J. W., *Probability and Random Processes with Applications to Signal Processing*, Prentice Hall, Upper Saddle River, NJ, 2002, pp. 421–423.

[19] Watkins, C. J. C. H., and Dayan, P., "Learning from Delayed Rewards," Ph.D. Thesis, Univ. of Cambridge, Cambridge, UK, 1989.

[20] Simsek, O., and Barto, A. G., "An Intrinsic Reward Mechanism for Efficient Exploration," *ACM International Conference Proceeding Series*, Pittsburgh, PA, 2006.

[21] Singh, S., Jaakkola, T., Littman, M. L., and Szepesvari, C., "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms," *Machine Learning*, Vol. 38, No. 3, March 2000, pp. 287–308.
doi: 10.1023/A:1007678930559

[22] Si, J., Barto, A. G., Powell, W. B., and II, D. W., *Learning and Approximate Dynamic Programming*, Wiley-Interscience, New York, NY, 2004.

Kelley Cohen
Associate Editor