J Control Theory Appl 2011 9 (3) 431–439
DOI 10.1007/s11768-011-1012-4

Multiresolution state-space discretization for Q-Learning with pseudorandomized discretization

Amanda LAMPTON [1], John VALASEK [2], Mrinal KUMAR [3]

1.Systems Technology, Inc., 13766 S. Hawthorne Blvd, Hawthorne, CA 90250, U.S.A.;

2.Department of Aerospace Engineering, Texas A&M University, 3141 TAMU, College Station, TX 77843-3141, U.S.A.;

3.Department of Mechanical & Aerospace Engineering, University of Florida, 306 MAE-A, Gainesville, FL 32611-6250, U.S.A.

Abstract: A multiresolution state-space discretization method with pseudorandom gridding is developed for the episodic unsupervised learning method of Q-learning. It is used as the learning agent for closed-loop control of morphing or highly reconfigurable systems. This paper develops a method whereby a state-space is adaptively discretized by progressively finer pseudorandom grids around the regions of interest within the state or learning space in an effort to break the Curse of Dimensionality. Utility of the method is demonstrated with application to the problem of a morphing airfoil, which is simulated by a computationally intensive computational fluid dynamics model. By setting the multiresolution method to define the region of interest by the goal the agent seeks, it is shown that this method with the pseudorandom grid can learn a specific goal within ± 0.001 while reducing the total number of state-action pairs needed to achieve this level of specificity to less than 3000.

Keywords: Reinforcement learning; Morphing; Random grid

1 Introduction

For the computational reinforcement learning 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.

Hierarchical reinforcement learning (HRL) seeks to split a complex learning problem into a hierarchy of subtasks or subgoals that can be learned individually. Determining those subgoals can prove to be a challenge. This type of learning is similar in thought to the multiresolution state-space discretization method developed in this paper. Reference [1] discusses an algorithm that combines Q-learning and a locally weighted learning method to select behavioral primitives and generate subgoals for the agent. Reference [2] identifies subgoals by partitioning local state transition graphs. Reference [3] develops the theory of quad-Q-learning. MAXQ is a popular HRL method that decomposes a learning problem into a hierarchy of subtasks to be learned using Q-learning. Thus, it is a hierarchical Q-learning algorithm. The method is first developed by Dietterich in [4].

A multiresolution state-space discretization method for Q-learning using a pseudorandom grid can improve the algorithm by gathering the most detailed information in and around regions of interest, namely, the goal. 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 [5]. An agent would therefore visit an infinite number of states using an infinite number of actions an infinite number of times. Consider 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 issue. This multiresolution method provides a means of learning the action-value function, $Q^{\pi}(s,a)$, for a fixed policy, π, in progressively finer detail. It seeks a compromise between the high rate of convergence of a coarse discretization, and the high level of detail of a fine discretization. This method essentially creates a series of smaller problems, similar to what is done in HRL, centered around a region of interest. This method was first introduced in [6] and is extended to having pseudorandom gridding to further break the curse of dimensionality.

The method mimics the natural tendency of people and animals to learn the broader goal before focusing on more specific goals within the same space. This method is applied to the morphing airfoil architecture developed in [6–9]. Reinforcement learning is used to learn the commands that produce the optimal shape based on airfoil lift coefficient goal. The levels of discretization of the state-space must be tuned such that good convergence and attention to detail is achieved. The contribution of this paper is to develop a new

Received 11 January 2011; revised 1 April 2011.
This work was partly supported by the Air Force Office of Scientific Research, USAF (No.FA9550-08-1-0038), and the National Science Foundation under a Graduate Research Fellowship.

discretization method that allows the learning to converge quickly while still maintaining a high level of detail around regions of interest in the environment.

This paper is organized as follows. Section 2 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. Sections 3 and 4 describe the pseudorandom grid and the multiresolution state-space discretization method with the pseudorandom gridding, respectively. Section 5 describes the policy comparison stopping criteria. Section 6 describes the airfoil model and how the airfoil problem is cast as a reinforcement learning problem. Section 7 takes the fully developed multiresolution state-space discretization method with pseudorandom gridding and the policy comparison stopping criteria, applies it to the morphing airfoil, and interprets a numerical example generated from it. Finally, conclusions are drawn from the numerical example in Section 8.

2 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 [10].

The agent and environment interact continually in 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, $r_{t+1} \in \mathbb{R}$, 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 $s_t = s$ [10]. The agent seeks to maximize the reward it receives or, more formally, its expected return, R_t.

In this paper, and for many reinforcement learning problems, it is assumed that the problems can be modeled as Markov decision processes (MDPs) and cast in the reinforcement learning problem framework. 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.

There are a number of ways to solve for the value or action-value functions that record the knowledge learned from reinforcement learning problems. Three basic solution methods are dynamic programming (DP), Monte Carlo methods, and temporal difference (TD) learning [10]. TD learning can be thought of as a combination of ideas from both DP and Monte Carlo [10]. 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. TD learning methods include TD prediction, Sarsa, Q-learning, actor-critic methods, etc.

The algorithm used here is one-step Q-learning, which is a common off-policy TD control algorithm. In its simplest form, it is a modified version of equation (1) and is defined by

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

The Q-learning algorithm is illustrated as follows [10]:

```
1) Initialize $Q(s, a)$ arbitrarily.
2) Repeat (for each episode).
  a) Initialize $s$;
  b) Repeat (for each step of the episode):
    . Choose $a$ from $s$ using policy derived from $Q(s, a)$ (e.g., $\varepsilon$-greedy policy),
    . Take action $a$, observe $r$, $s_{t+1}$,
    . $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$,
    . $s \leftarrow s_{t+1}$;
  c) until $s$ is terminal;
3) Return $Q(s, a)$.
```

The agent learns the greedy policy, and 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. Another 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 $1 - \varepsilon$, otherwise a random action is selected.

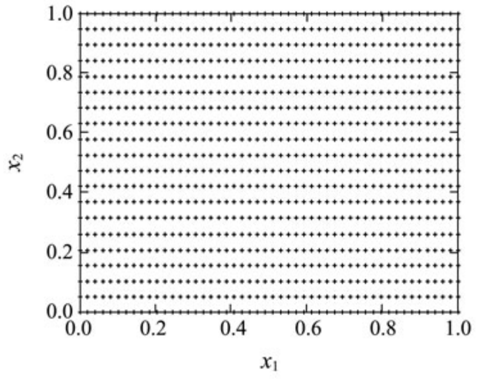
3 Staggered grid generation

It is well known that orthodox rectangular grid-based discretization suffers from the curse of dimensionality. The current problem of Q-learning presents a similar challenge because it is desired to uniformly place nodes in high-dimensional spaces. This should be achieved without causing an explosion in the number of nodes required to obtain efficient domain coverage. One way to fulfill this objective is through the use of random number generators (Monte Carlo methods). A random number generator typically creates a uniformly distributed sample that covers the domain of interest. Another technique, which is especially popular in recent times, is the use of pseudorandom numbers. These various methods of domain discretization are illustrated in

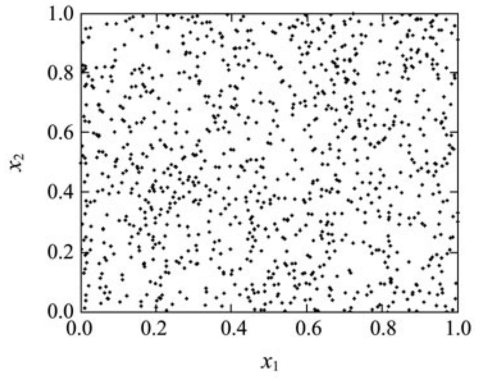
Fig. 1.

Also called quasirandom numbers, pseudorandom numbers have properties similar to uniformly distributed random samples but are generated algorithmically (as opposed to a random number generator). They have been widely used to evaluate high-dimensional integrals and solve partial differential equations. They provide distinct benefits over both orthodox lattice like grids and purely random Monte Carlo methods. Compared to the former, they provide an easy way of breaking the curse of dimensionality because no meshing among nodes is required. This is also true for Monte Carlo methods (compare Fig. 1 (a) to Figs. 1 (b)–(c)). Compared to Monte Carlo methods, pseudorandom grids provide at least two distinct benefits. First, as opposed to purely random numbers, they can be generated algorithmically, which is preferable for generating repeatable node distributions. Second, since they are generated algorithmically, deterministic performance measures can be given for algorithms based on them. This is different from Monte Carlo methods, for which only stochastic (average) performance measures can be described.

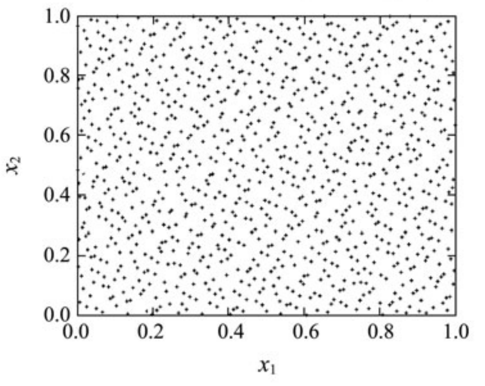
There exists a slew of algorithms to generate sequences of quasirandom numbers, e.g., Halton's algorithm, Fauvre's method, Sobol sequence, etc. In this work, Halton's sequence is used. For details about this popular algorithm, the interested reader is directed to Niederreiter [11]. Figs. 1 (c) and (d) show Halton pseudorandom samples in two- and three-dimensional spaces.



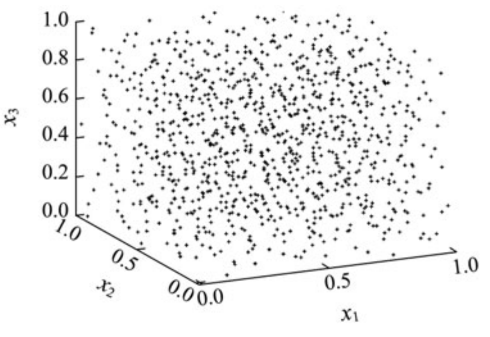
(a) An orthodox, grid-based discretization of a two-dimensional domain (50×20 grid)



(b) Discretization with purely random numbers drawn from a uniform distribution



(c) 1000 pseudo-random numbers generated using Halton's algorithm in two dimensions



(d) 1000 pseudo-random numbers generated using Halton's algorithm in three dimensions

Fig. 1 Various methods of domain discretization.

To implement Q-learning, it is required to determine neighbors for each node in the domain. The actions, $A(s)$, for the Q-learning agent are defined by movement from a node to one of its neighboring nodes. Determining the neighbors can be done in several ways. The simple techniques used in this work for determining neighbors are the following:

Nearest directional neighbors An educated search involves looking for neighbors along each dimension of the domain. As in rectangular grid arrangements, some neighbors may not exist (for example, in a two-dimensional grid, there is no node to the 'right' of a corner node with coordinates $(x_{\max}, \cdot)$). In the current work, the following criteria was used to extract directional neighbors: for a node $\boldsymbol{x}_i \in \mathbb{R}^N$, node $\boldsymbol{x}_j \in \mathbb{R}^N$ is its neighbor to the 'right' in dimension k if:

$$\begin{cases} \boldsymbol{v}_{ij}^k \geqslant 0, \\ \boldsymbol{v}_{ij}^k = \min\limits_{q \neq i}\{\boldsymbol{v}_{iq}^k | \boldsymbol{v}_{iq}^k \geqslant 0\}, \\ \boldsymbol{v}_{ij}^k = \max\limits_{n=1,\dots,N} |\boldsymbol{v}_{ij}^n|. \end{cases} \tag{2}$$

In the above equation, $\boldsymbol{v}_{ij}^k$ represents the kth component of the displacement vector between nodes i and j, i.e., $\boldsymbol{v}_{ij} = \boldsymbol{v}_j - \boldsymbol{v}_i$. Therefore, the first condition in equation (2) ensures that the node $\boldsymbol{x}_j$ lies to the 'right' of node $\boldsymbol{x}_i$ in dimension k. The second condition ensures that node $\boldsymbol{x}_j$ is closest to node $\boldsymbol{x}_i$ among all nodes lying to the right of $\boldsymbol{x}_i$ along dimension k. Note that only the kth component (and not the Euclidean distance) is considered to measure closeness. Finally, the third condition ensures that the kth component of the displacement vector $\boldsymbol{v}_{ij}$ has the greatest magnitude among components of $\boldsymbol{v}_{ij}$. This final condition introduces the element of direction in the search and ensures that the neighbor $\boldsymbol{x}_j$ is indeed 'along the dimension k'. All three conditions enforced together result in the closest node to the right of $\boldsymbol{x}_i$ along dimension k. Along similar lines, the left neighbor of node $\boldsymbol{x}_i$ along dimension k is node $\boldsymbol{x}_{\mathrm{l}}$ if

$$\begin{cases} \boldsymbol{v}_{i\mathrm{l}}^{k} \leqslant 0, \\ \boldsymbol{v}_{i\mathrm{l}}^{k} = \max\limits_{q \neq i}\{\boldsymbol{v}_{iq}^{k} | \boldsymbol{v}_{iq}^{k} \leqslant 0\}, \\ \boldsymbol{v}_{i\mathrm{l}}^{k} = \max\limits_{n=1,\dots,N} |\boldsymbol{v}_{i\mathrm{l}}^{n}|. \end{cases} \tag{3}$$

Fig. 2 illustrates directional neighbors for a node distribution. Note that according to the first condition alone in equation (2), node 1 could be identified as both x_{right} and y_{right}. However, conditions 2 and 3 identify it as a node 'in the x direction' as opposed to a node in the y direction, and hence its label.

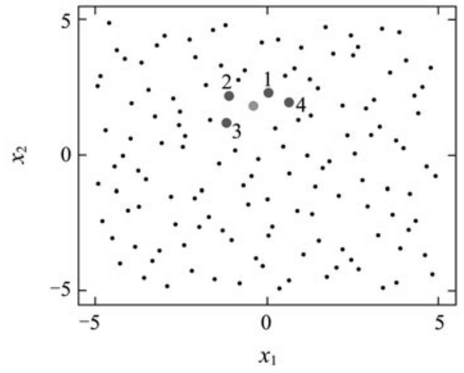


Fig. 2 Determining directional neighbors based on equations (2) and (3) in staggered node distributions.

4 Multiresolution state-space discretization (AAG) for pseudorandom discretization

This method was first developed for a uniform discretization and is described in detail in [6]. Here, it is extended to the pseudorandom grid described in the previous section. Discretizing a state-space for learning is beneficial in that it creates a finite number of state-action pairs the agent must visit. Discretizing using a pseudorandom grid is beneficial in that it further breaks the curse of dimensionality. Generally, as the number of state-action pairs decreases, the rate of convergence increases [8]. However, fewer state-action pairs capture less detail of the environment. As noted in passing in Section 3, the agent moves from node to neighboring node, and that movement to either a directional neighbor or to a nearest Euclidean distance neighbor comprises the actions of the agent. It is entirely possible that the goal the agent is seeking, or any other region of interest, does not lie on a node. This necessitates adding a range to the goal that encompasses one or more of the nodes in the state-space. These nodes within the goal range are pseudogoals (Fig. 3 (a)).

As the agent explores the coarsely discretized state-space and garners rewards, it also notes the location of the pseudogoals. Once learning on the current discretization has converged, the area surrounding and encompassing the pseudogoals is rediscretized to a finer resolution using Halton's sequence to generate a pseudorandom grid in the region of interest, and the new neighbors are determined. A new, smaller range is defined for the goal and learning begins anew in the smaller state-space. Fig. 3 (b) shows the rediscretization of the state-space in two dimensions, although this method is extensible to N-dimensions.

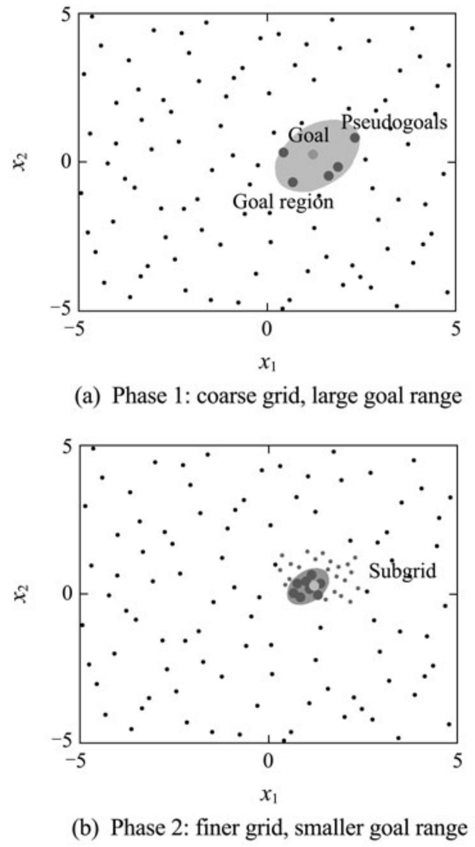


Fig. 3 Multiresolution state-space discretization.

Currently, the number of nodes in each level of discretization is preset by the user before learning commences and is another parameter that must be tuned to ensure that the learning converges. The total number of nodes is therefore simply

$$N_{V_N} = \sum_{j=1}^{M} v_j, \tag{4}$$

where N_{V_N} is the total number of nodes or vertices in N-dimensions, j is the resolution of the discretization in which 1 is the coarsest or first level, M is the finest or last level, and v_j is the number of nodes for the jth level of discretization. This leads to

$$N_N = \sum_{j=1}^{M} 2Nv_j \tag{5}$$

state-action pairs if neighbors are defined by direction.

This method reduces a learning problem to a series of smaller learning problems with relatively few state-action pairs, on the order of several orders of less magnitude. Rather than one large problem that will take a great deal of time to converge, there are several quickly converging smaller problems with pseudorandom discretization.

5 Policy comparison

The multiresolution discretization method provides a means of learning the action-value function, $Q^{\pi}(s,a)$, for a fixed policy, π, in progressively finer detail. Rather than blindly allowing the agent to learn for the entire number of user defined episodes, stopping criteria based on the learned policy are introduced.

In Q-learning, all of the information is stored in the form of the action-value matrix, often in the form of a table. The learned greedy policy itself is not represented explicitly or with any sort of model. However, the policy and associated value function can be easily extracted from the action-value function in a few simple steps. There exists a simple relationship between the action-value function and the greedy policy, namely,

$$\pi(s) = \arg\max_{a} Q(s,a), \tag{6}$$

where $\pi(s)$ is the action associated with the maximum preference over the set of actions for the state.

In addition, a representation of the value function can then be easily calculated:

$$V(s) = \max_{a} Q(s,a) \tag{7}$$

for the tabular action-value function. This relationship will be used for visual analysis in later sections.

Two stopping criteria are added to the Q-learning algorithm and form the third and final addition that makes up the new overall algorithm developed here. These two criteria are a direct policy comparison and a performance-based policy comparison, which are periodically applied to the learned action-value function to determine if the action-value function has converged to a usable data set, where a usable data set refers to an action-value function that enables the agent to successfully navigate from any initial state to a goal state. Both criteria use the relationships in equation (6) for the tabular action-value function. Both stopping criteria introduced in this section must be met for learning at the current level of discretization to be terminated, otherwise learning continues. These criteria are described in the following sections.

5.1 Direct policy comparison

The direct policy comparison stopping criterion is a simple and expedient way to track the change in the policy extracted from an action-value function as that function evolves during learning. This policy comparison is carried out in a short series of steps:

1) Pause learning after n episodes.

2) Extract current greedy policy, $\pi_i(s)$, from action-value function, where i is the number of elapsed episodes divided by n.

3) Directly compare $\pi_i(s)$ and $\pi_{i-1}(s)$.

4) If change in policy, $\Delta\pi$, is $< \varepsilon$, then stopping criterion #1 is achieved.

Here, ε is a small number and is usually set as 5% for this research, and $\Delta\pi$ is essentially the fraction of states that are different between π_i and π_{i-1}. Also, after learning is initialized and the first set of n episodes elapse, the policy, $\pi_1(s)$, is extracted and stored only if there is no $\pi_0(s)$. This comparison method is fully utilized starting after the second set of n elapsed episodes.

It is entirely possible that a lack of change in the extracted policy is not an indication that the action-value function has converged to a usable function. It could simply be a momentary aberration and would begin to change again if another n episodes is allowed. To prevent this from occurring, a second stopping criterion is introduced.

5.2 Performance-based policy comparison – Monte Carlo simulation

The second stopping criterion implemented is based on the performance of the current policy. During the pause in learning after every n episodes, the policy comparison is conducted and then this performance-based policy comparison. The performance-based policy comparison measures 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 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 outermost boundary of the state-space, the boundary of the current level of discretization, or gets 'lost' and wanders around the state-space, which is identified when the maximum number of allowed actions is reached without encountering a goal state. 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 using equation (8).

$$\%\ \text{Success} = \frac{\#\ \text{of Successes}}{\text{Total}\ \#\ \text{of Simulations}}. \tag{8}$$

When this success percentage is above some threshold, usually 98% in this research, the second stopping criterion is satisfied. Both first and second stopping criteria 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.

6 Simple reconfigurable system example – morphing airfoil

When considering a reconfigurable air vehicle, the complexity of the reconfiguration can range from changing a few parameters, such as wing dihedral, wing sweep, wing span, the airfoil itself, etc., to a fully articulated wing, body, and tail much like a bird's. Aircraft are usually designed for very specific purposes. Many current approaches involve optimizing the vehicle shape for several flight phases, applying an optimal control technique of one form or another, and determining an actuation scheme to enable the shape change. Some of the problems that arise with this approach are that there are only a handful of optimal shapes and the optimized controller is specific to the initial and final shapes. Should the optimal configuration be redesigned, that would require that the shape change controller be redesigned as well. Machine learning and in particular the algorithm developed in Section 4 of this paper is a candidate approach to avoid these problems.

The first application of this algorithm to a reconfigurable system is a simple airfoil as seen in Fig. 4. Rather than finding a couple optimal shapes that meet some criteria and a separate optimal controller to maneuver from one shape to the other, the airfoil is cast as a reinforcement learning problem in which the full spectrum of shapes that meet the flight phase criteria are learned as well as the local transitions that guide the shape from any initial shape to a shape that meets the requirements.

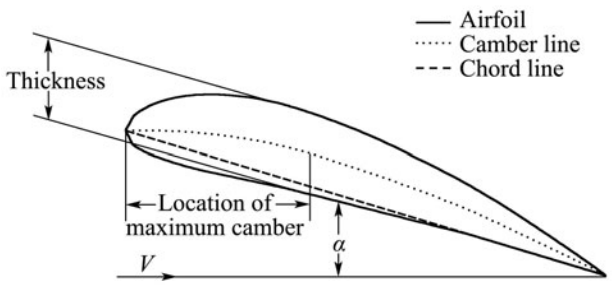


Fig. 4 Representative airfoil.

Section 6.1 discusses the airfoil model to be cast as the environment of the reinforcement learning problem. Then, the full reconfigurable airfoil reinforcement learning problem is described followed by analysis of the algorithm described above when applied to reconfigurable airfoil reinforcement learning problem.

6.1 Airfoil model

The airfoil is modeled by a computational fluid dynamics (CFD) code using a constant strength doublet panel method. This model calculates the aerodynamic properties of an airfoil given a set of four inputs: airfoil thickness, airfoil camber, location of maximum camber, and airfoil angle-of-attack. Setting these four parameters as inputs to the model allow for quick calculation of the aerodynamic properties of many different airfoils or as a single airfoil changes shape. The full development, validation, and verification of this model can be found in [7].

6.2 Airfoil cast as a reinforcement learning problem

In the case of the airfoil, there are no dynamics involved in the shape change at this point in the problem development. Thus, each commanded change in airfoil shape yields an immediate response. This formulation allows for better focus on the actual choice in shape.

The state variables are chosen such that the aerodynamic properties of the airfoil are adequately exploited. The CFD model itself constitutes the environment with which the agent interacts. Thus, there are four possible interdependent parameters, consisting of the four inputs to the CFD model, which constitute the state of the agent within the environment. The reinforcement learning problem can be cast such that any combination of the four parameters forms the state, while the others are just held constant in the background.

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 Section 3, so these actions include incremental changes in the shape parameters of the airfoil defined by the neighboring nodes. Thus, the agent is effectively restricted to movement between adjacent nodes. An example of admissible action in this context is the following. The agent chooses to move in the x_1-dimension from node $\boldsymbol{x}_i$ in the two-dimensional problem. The two possible actions in the x_1-dimension are defined as follows:

$$\begin{cases} A_{11} \equiv x_i \rightarrow x_1, \\ A_{1j} \equiv x_i \rightarrow x_j, \end{cases} \tag{9}$$

where A_{11} is the action from the current node to the node to the left along the x_1-dimension, and A_{1j} is the action from the current node to the node to the right. The definitions of the x_i axes are defined in Table 1.

Table 1 Morphing airfoil axis definitions.

x_i	Definition
x_1	Thickness/%
x_2	Camber/%

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 presented here has a goal defined by the airfoil lift coefficient, c_{l}. Those nodes whose state yield aerodynamic coefficients, c, from the CFD model that lie in the goal range defined by $g - g_{\mathrm{r}} \leqslant c \leqslant g + g_{\mathrm{r}}$ form the pseudogoals of the problem.

The reward structure for this problem is the function defined by equation (10). The limits that define the bounds of the state-space are listed in Table 2.

$$r = |g - c_{n-1}| - |g - c_n|, \tag{10}$$

where r is the reward, g is the goal, and c is the metric or aerodynamic coefficient.

Table 2 Morphing airfoil parameter limits.

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

7 Numerical example

The purpose of the numerical example is to demonstrate the learning performance of the Q-learning with a pseudorandom state-space discretization when integrated with the aerodynamic model. The agent follows a 100% exploration policy for this example. The agent is to learn the action-value function with multiple levels of discretization in which the discretization is a randomized grid and using the stopping criteria. The number of nodes in each finer discretization is preset for the problem. The number of levels of discretization or resolution as defined earlier is M. The parameters for this problem are listed in Table 3.

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 the example are defined by the four directional neighbors determined using equations (2) and (3).

During learning, knowledge of the action-value function is carried over between discretizations and the reward structure defined by equation (10) is used. When learning is paused, the action-value function is evaluated using the policy comparison stopping criteria developed in Section 5. The learning is analyzed by evaluating the Monte Carlo simulation performance results and the final value function and associated greedy policy.

Table 3 Airfoil parameters and constants.

Parameter	Value
g_{f}	0.20
M	3
Goal g	$c_{\mathrm{l}} = 0.3$
Initial range g_{r_1}	0.025
α	0.01
γ	0.7
Chord	1 m
Angle-of-attack	2.0°
Episodes	5000
Number of nodes	$[v_1\ v_2\ v_3] = [181\ 359\ 2003]$

7.1 Monte Carlo simulation

Fig. 5 shows the results of the Monte Carlo simulation performance analysis and the final distribution of states for this problem. Each level of discretization is again allowed a possible 5000 episodes. The final range for the goal in this problem is 0.001. Learning on each level of discretization for this case is terminated when there is less than 5% change in the policy extracted from the action-value function in equation (6) and the policy performance analysis yields $>98\%$ success.

The figure shows that the first discretization reaches $>98\%$ within 200 episodes, and the learning is terminated after 800 episodes. The agent still has learning to do after the first 200 episodes because the direct policy comparison stopping criterion has not yet been met. The second discretization converges, reaches $>98\%$ success, and terminates learning within 800 episodes. The third discretization reaches $>98\%$ success within 1400 episodes. Based on these learned data, only 3000 episodes were needed of the 15000 allowed episodes for learning to converge using the pseudorandom grid, which is an 80% reduction. Also, Fig. 5 (b) shows the concentration of nodes near 2.4% camber. Discretizing the entire domain to the density in this area would result in a pseudorandom grid with over 19000 nodes. The 2543 nodes in this figure is a 86.7% reduction.

The reason why there is such a sharp decline in success at the start of the third level of discretization is that the policy in the region of interest is learned for the second-level discretization. When the region is rediscretized with a pseudorandom grid the final region, the action-values for the new states are interpolated from the previous discretization. For this example, the goal of $c_{\mathrm{l}} = 0.3$ lies between vertices of the second-level discretization, although it does not directly bisect the distance between adjacent vertices. When the region is rediscretized and the action-values are interpolated, this is not taken into account because the agent is not aware of this detail. The resulting policy after interpolation reflects what was learned at the second level of discretization and must be modified by additional learning to take into account the goal in relation to the states. This results in a drop in success rate until the interpolated action-values for the new states are directly reinforced by the agent for the new discretization.

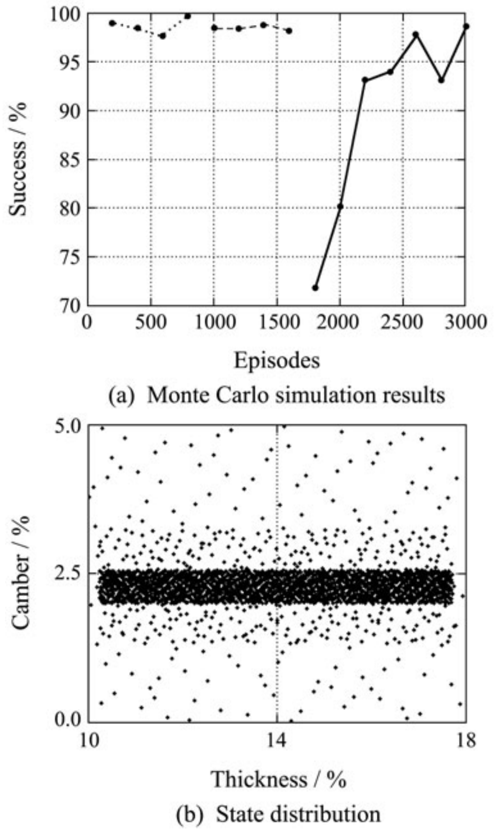
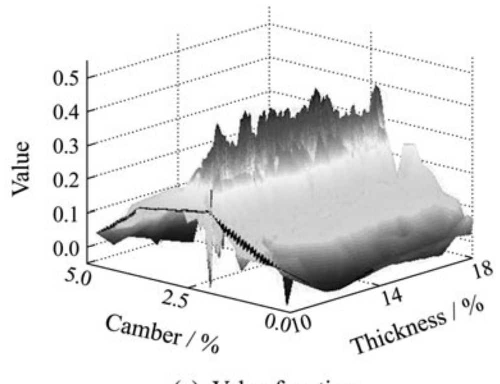


Fig. 5 Final simulation results of learned knowledge of airfoil.

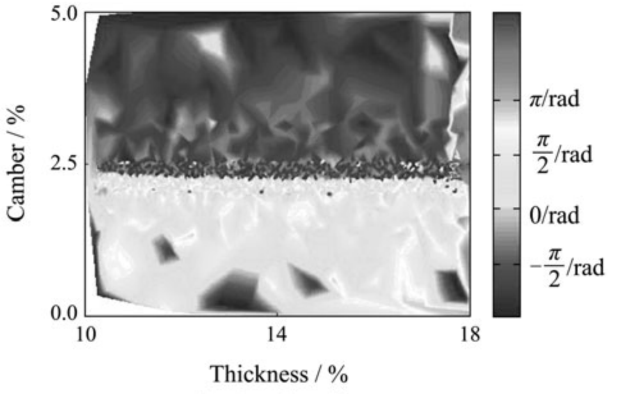
7.2 Value function and policy analysis

The following figure shows the greedy policy and value function based on the final action-value function and are determined using equations (6) and (7), respectively. The actions for the policy are color coded for visual clarity and are defined by the angle between the x_1-axis and the line made by the current state, s, and the subsequent state, s', which occurs given the greedy action.

Fig. 6 illustrates the value function and greedy policy based on the final action-value function. The sudden increase in value function as the camber approaches 2.4% is a result of learning on the second level of discretization. Notice the function appears more refined in this region. The ridge in the middle of the function is effectively the goal of $c_{\mathrm{l}} = 0.3$ and where the reward function approaches 0. This ridge is reflected in the graphic of the greedy policy. This figure shows that change in camber has the greatest influence on airfoil lift coefficient. The vagaries in the directions indicated by the color distribution in Fig. 6 (b) is a result of the pseudorandom grid and how the directional neighbors are determined. This figure and the other analyses show that this approach is successful and is a good next step for AAG.



(a) Value function



(b) Greedy policy

Fig. 6 Representation of learned knowledge of airfoil.

8 Conclusions

The results show that the learning for the multiresolution method with pseudorandom gridding reaches a 98% success rate or greater within 200 episodes for the coarsest discretization and within 1400 episodes for the finest discretization. Conceptually, a problem with the full state-space discretized at the finest level would take many thousands or tens of thousands of episodes more to reach this level of convergence. This method is successful in greatly reducing the time for convergence, increasing the rate of convergence, and achieving a goal with the very small range of 0.001 with less than 3000 states. This method essentially reduced the larger problem by almost 90%, making it a much more computationally tractable learning problem. Additionally, the value function and policy show that learning for the example considered, the lift coefficient is dominated by the camber.

Acknowledgements

This material is based upon work supported in part by the U.S. Air Force Office of Scientific Research under contract FA9550-08-1-0038, under technical monitor Dr. Fariba FAHROO, and by the National Science Foundation under a Graduate Research Fellowship. This support is gratefully acknowledged by the authors. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Air Force or the National Science Foundation.

References

[1] D. C. Bentivegna, C. G. Atkeson, G. Cheng. Learning to select primitives and generate sub-goals from practice. *Proceedings of the IEEE/lRSJ International Conference on Intelligent Robots and Systems*, New York: IEEE, 2003: 946 – 953.

[2] O. Simsek, A. P.Wolfe, A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proceedings of the 22nd International Conference on Machine Learning*, New York: ACM, 2005: 816 – 823.

[3] C. Clausen, H. Wechsler. Quad-*Q*-learning. *IEEE Transactions on Neural Networks*, 2000, 11(2): 279 – 294.

[4] T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. *Proceedings of the 15th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1998: 118 – 126.

[5] C. J. C. H. Watkins, P. Dayan. *Learning from Delayed Rewards*. Ph.D. thesis. Cambridge, U.K.: University of Cambridge, 1989.

[6] A. Lampton. *Function Approximation and Discretization Methods for Reinforcement Learning of Highly Reconfigurable Vehicles*. Ph.D. thesis. College Station, TX: Texas A&M University, 2009.

[7] A. Lampton, A. Niksch, J. Valasek. Reinforcement learning of morphing airfoils with aerodynamic and structural effects. *Journal of Aerospace Computing, Information, and Communication*, 2009, 6(1): 30 – 50.

[8] A. Lampton, A. Niksch, J. Valasek. Reinforcement learning of a morphing airfoil-policy and discrete learning analysis. *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008: No.AIAA-2008-7281.

[9] A. Lampton, A. Niksch, J. Valasek. Morphing airfoil with reinforcement learning of four shape changing parameters. *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Honolulu, HI, 2008: No. AIAA-2008-7282.

[10] R. Sutton, A. Barto. *Reinforcement Learning – An Introduction*. Cambridge: MIT Press, 1998.

[11] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM, 1992.

Amanda LAMPTON joined Systems Technology, Inc. in June, 2010. Her main areas of interest are flight mechanics and control, intelligent control, and autonomous systems. As a graduate student, she studied utilizing learning algorithms to solve the shape control problem of a reconfigurable or morphing air vehicle cast as a reinforcement learning problem. The majority of this work was funded by a National Science Foundation Graduate Research Fellowship. She developed the methodology to the point that funding was secured from the Air Force Office of Scientific Research (AFOSR) for further development. She served as student technical lead on this project for the remainder of her graduate career and her post-doctoral research. Dr. Lampton has extensive experience in designing flight controllers for a myriad of problems including the morphing aircraft problem, aerial refueling tasks, and high performance aircraft regulators and autopilot tasks using a variety of techniques. Amanda serves on both the AIAA Guidance, Navigation, and Control Technical Committee and the AIAA Intelligent Systems Technical Committee. She earned her B.S., M.S., and Ph.D. degrees in Aerospace Engineering at Texas A&M University in 2004, 2006, and 2009, respectively. She is a member of the American Institute of Aeronautics and Astronautics (AIAA) and the Institute of Electrical and Electronics Engineers (IEEE).

John VALASEK is Director, Vehicle Systems & Control Laboratory and Professor of Aerospace Engineering at Texas A&M University. His research focuses on bridging the gap between computer science and aerospace engineering, encompassing machine learning and multiagent systems, intelligent autonomous control, vision-based navigation systems, fault tolerant adaptive control, and cockpit systems and displays. John was previously a flight control engineer for the Northrop Corporation, Aircraft Division where he worked in the Flight Controls Research Group, and on the AGM-137 Tri-Services Standoff Attack Missile (TSSAM) program. He was also a summer faculty researcher at NASA Langley in 1996 and an AFOSR summer faculty research fellow in the Air Vehicles Directorate, Air Force Research Laboratory in 1997. John has served as Chair of Committee to 32 completed graduate degrees, and his students have won national and regional student research competitions in topics ranging from aircraft design to smart materials to computational intelligence. John is an associate editor of the Journal of Guidance, Control, and Dynamics, and current member of the AIAA Guidance, Navigation, and Control Technical Committee; AIAA Intelligent Systems Technical Committee; and the IEEE Technical Committee on Intelligent Learning in Control Systems. John earned his B.S. degree in Aerospace Engineering from California State Polytechnic University, Pomona in 1986, M.S. degree with honors, and Ph.D. in Aerospace Engineering from the University of Kansas, in 1990 and 1995, respectively. He is an Associate Fellow of AIAA and a Senior Member of IEEE.

Mrinal KUMAR received his B.Tech. degree from Indian Institute of Technology, Kanpur in 2004, and Ph.D. from Texas A&M University in 2009, both in Aerospace Engineering. He is currently a member of the faculty as an assistant professor in the Department of Mechanical and Aerospace Engineering at University of Florida, Gainesville. His current research interests include uncertainty quantification using spectral methods and design of randomized algorithms for large scale engineering problems.