Function Approximation via Tile Coding: Automating Parameter Choice

Alexander A. Sherstov and Peter Stone

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 USA
{sherstov, pstone}@cs.utexas.edu

Abstract. Reinforcement learning (RL) is a powerful abstraction of sequential decision making that has an established theoretical foundation and has proven effective in a variety of small, simulated domains. The success of RL on real-world problems with large, often continuous state and action spaces hinges on effective *function approximation.* Of the many function approximation schemes proposed, *tile coding* strikes an empirically successful balance among representational power, computational cost, and ease of use and has been widely adopted in recent RL work. This paper demonstrates that the performance of tile coding is quite sensitive to parameterization. We present detailed experiments that isolate the effects of parameter choices and provide guidance to their setting. We further illustrate that *no single parameterization* achieves the best performance throughout the learning curve, and contribute an *automated* technique for adjusting tile-coding parameters online. Our experimental findings confirm the superiority of adaptive parameterization to fixed settings. This work aims to automate the choice of approximation scheme not only on a problem basis but also throughout the learning process, eliminating the need for a substantial tuning effort.

1 Introduction

Temporal-difference reinforcement learning (RL) is a powerful machine-learning methodology that has an established theoretical foundation and has proven effective in a variety of small, simulated domains. The application of RL to practical problems, however, is problematic due to their large, often continuous state-action spaces. Recently RL has been successfully applied to larger problems, including domains with continuous state-action spaces. The success of RL in such cases critically depends on effective *function approximation*, a facility for representing the value function concisely at infinitely many points and generalizing value estimates to unseen regions of the state-action space.

A variety of function approximation methods for RL have been proposed, including simple discretization, radial basis functions, instance- and case-based approximators, and neural networks [1]. These methods trade off representational power, computational cost, and ease of use. *Tile coding* [2] is a linear function-approximation method that strikes an empirically successful balance along these dimensions and has been widely adopted in recent work [3, 1, 4–6]. The success of tile coding in practice depends in large part on parameter choices. We are not aware of any detailed studies of the effects of parameters in tile coding, an omission we set out to address.

This paper makes two chief contributions. First, we present a controlled empirical study of the effects of parameters in tile coding. While it is natural to expect the right parameterization to depend on the *problem* at hand, we additionally demonstrate that no single parameterization achieves the best performance on the *same* problem throughout the learning curve. Our analysis isolates the causes of these phenomena. Second, this paper contributes an automated scheme for adjusting tile-coding parameters online. We demonstrate the superiority of online parameter adjustment to any fixed setting.

Our work on adaptive parameterization in tile coding automates the choice of an appropriate approximation scheme for any given RL *problem* and *learning stage*. The designer need only specify a parameter range, leaving it up to the algorithm to determine the right settings throughout execution. Viewed differently, our work unifies fixed approximation schemes into a more powerful and generic scheme. We validate our insights empirically in the context of RL, arguably the most realistic and successful abstraction of sequential decision making to date.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of RL and describes tile coding and our testbed domain. Experimental results in multiple learning settings and an accompanying analysis are presented in Sections 3 and 4, respectively. Section 5 builds on those findings to propose an automated parameter-adjustment scheme and demonstrates its effectiveness empirically. Section 6 concludes with a summary.

2 Preliminaries

This section introduces reinforcement learning (RL) and tile coding and describes the testbed domain used in our experiments.

2.1 Reinforcement learning

In RL [2], a *learner* is placed in a poorly understood, possibly stochastic and nonstationary *environment*. The learner interacts with the environment at discrete time steps. At every time step, the learner can observe and change the environment's *state* through its *actions*. In addition to state changes, the environment responds to the learner's actions with a *reward*, a scalar quantity that represents the immediate utility of taking a given action in a given state. The learner's objective is to develop a *policy* (a mapping from states to actions) that maximizes its long-term return.

Formally, an RL problem is given by the quadruple $\langle \mathcal{S}, \mathcal{A}, t, r \rangle$, where $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $t : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is a *transition function* that specifies the probability of observing a certain state after taking a given action in a given state; and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function* that specifies the expected reward upon taking a given action in a given state. Given a stream of rewards $r_0, r_1, r_2, \ldots$, the associated return is defined as $\sum_{i=0}^{\infty} \gamma^i r_i$, where $0 \leq \gamma \leq 1$ is the *discount factor*. The learner experiences the world as a sequence of states, actions, and rewards, with no prior knowledge of the functions t and r. A practical vehicle for learning in this setting is the *value function* $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that yields the expected long-term return obtained by taking a certain action in a given state and following policy π thereafter. The widely used Q-learning algorithm [7] maintains and iteratively updates an approximation to the Q-value function of the optimal policy.

2.2 Tile coding

In practical applications of RL, states and actions are defined by continuous parameters such as distances and voltages. As a result, the sets $\mathcal{S}$ and $\mathcal{A}$ are typically large or infinite, and learning the value function requires some form of *function approximation*. In *tile coding*, the variable space is partitioned into *tiles*. Any such partition is called a *tiling*. The method uses several overlapping tilings and for each tiling, maintains the weights of its tiles. The approximate value of a given point is found by summing the weights of the tiles, one per tiling, in which it is contained. Given a training example, the method adjusts the weights of the involved tiles by the same amount to reduce the error on the example.

Figure 1 illustrates tile coding as it is used in this paper. The variable space consists of a single continuous variable x. The tiles are all the same width and adjacent tilings are offset by the same amount, the type of tiling organization we refer to as *canonical*. Figure 1 also illustrates the computation of value estimates. A tiling organization such as those in Figure 1 is given by tile width w and the number of tilings t. The ratio w/t is the *resolution* of a tiling organization. Speaking of tiling organizations that provide the same resolution, we refer to the number of tilings as the *breadth of generalization* since tiling organizations with more tilings generalize more broadly. This happens because the span of the tiles activated by an update grows with the number of tilings. A degenerate form of tile coding is straight discretization (the organization with a single tiling in Figure 1), which does not generalize across tile boundaries.

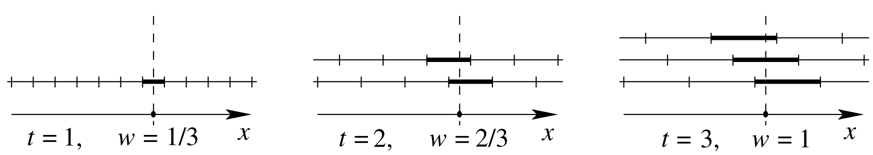


Fig. 1: One-, two-, and three-tiling canonical organizations with the same resolution $r = 1/3$. The number of tilings t and tile width w are specified for each organization. In each case, the weights of the highlighted tiles are summed to obtain the value estimate for the indicated point.

Note that tile coding is a *piecewise constant* approximation scheme: for any assignment of the tile weights, there will be actions within resolution r of each other that map to the same set of tiles and share the same value estimate. When pondering an action choice in this setting, our RL algorithm picks the middle action. While tile coding does not support *truly* continuous learning, its generalization capability makes it far superior to straight discretization. Finer distinctions can always be learned by increasing the resolution r. An initial result regarding tiling organizations is (a proof sketch is in the appendix):

Theorem 1. *For every $m, n \geq 1$, the sets of functions representable by m- and n-tiling canonical univariate organizations with the same resolution are identical.*

For example, the three organizations in Figure 1 are functionally equivalent. Despite this representational equivalence and identical *asymptotic* performance, tiling organizations with more tilings generalize more broadly and perform differently on RL tasks. This paper assumes a fixed resolution r and studies the role of generalization breadth t as a parameter. Our work seeks to identify how varying the breadth of generalization—while preserving representational equivalence—affects performance.

2.3 Testbed domain

Our testbed domain is a grid world, shown along with an optimal policy in Figure 2. Two locations of the grid world are designated as "start" and "goal," with the learner's objective being to navigate from the start cell to the goal cell. Another type of cell is a wall that the learner cannot pass through. Finally, certain cells are designated as an abyss. This grid world task is episodic, ending with the learner falling into the abyss ("stepping off the cliff") or successfully entering the goal state. The state variables are the cell coordinates x and y (the start state is at the origin).

The learner's actions are of the form (d, p), where $d \in \{$NORTH, SOUTH, EAST, WEST$\}$ is an intended direction of travel and p is a real-valued number between 0 and 1. The learner moves in the requested direction with probability $F(x, y, p)$, and in one of the three other directions with probability $(1 - F(x, y, p))/3$. Moves into walls and off the edge of the grid world result in no change of cell. F is a cell-dependent function that maps p to $[0.5, 1]$. The two "extreme" F functions are shown in Figure 3, and the F functions for all other cells are successive interpolations between these two.[1] This design of F was intended to ensure continuity as well as multiple local maxima and minima. To illustrate, consider choosing an action in a cell governed by the solid F curve in Figure 3. A choice of $p \approx 0.78$ guarantees a successful move in the requested direction. A choice of $p \approx 0.93$ moves the learner in the requested direction with probability 0.5 and in each of the other three directions with probability ≈ 0.17.

Our experiments use two different reward functions to guide the learner to the goal, an "informative" one with -1 assigned on every nonterminal transition, -100 on stepping off the cliff, and $+100$ on reaching the goal cell; and another, "uninformative" reward function with zero reward assigned on all transitions except the one to the goal cell $(+100)$. Because of the discount parameter $\gamma = 0.99 < 1$, the optimal policy is the same under both functions.

The learner initially has no information regarding t, r, or any of the F's. Thus, the challenge is to identify, in each cell, the right direction of travel and the p value

[1] The exact functional form is $F(x, y, p) = \frac{1}{3} w(p, x, y) \sin(4\pi w(x, y, p)) + \frac{19}{24}$, where $w(\cdot)$ is a warping function that applies a different monotonic transformation of the p range for each cell. As a result, the optimum p value is different for every cell. As the cells are traversed in row-major order, the F curve gradually transforms from one extreme in Figure 3 to the other.

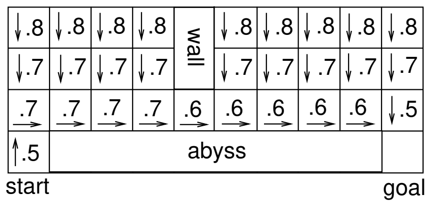


Fig. 2: The grid world map and optimal policy.

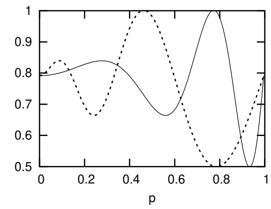


Fig. 3: The two extreme $F(p)$ functions.

that maximizes the probability of this move. We use tile coding in the p variable to approximate the value function for every distinct setting of (x, y, d). Every (x, y, d) triple enjoys a dedicated set of tiles, so there is no generalization across cell boundaries or directions of travel.

3 Initial Empirical Results

This section presents empirical results in three scenarios illustrating the effects of the breadth of generalization on performance. The settings of generalization breadth compared are 1, 3, and 6 tilings, all reasonable choices given the target function curves in Figure 3. The resolution was fixed at 0.04, corresponding to 26, 10, and 6 tiles per tiling in the 1, 3, and 6 tiling cases, respectively.

All experiments in this paper used Q-learning with ϵ-greedy action selection. The parameter settings were: $\alpha = 0.1, \gamma = 0.99$, and $\epsilon = 0.05$, except where indicated otherwise. The Q-value estimates were initialized to 0 on all runs. The metric in all experiments was the value of the start state under the best policy discovered so far (as a percentage of optimal), as determined by an external policy-evaluation module. This model-based evaluation module (value iteration) was unrelated to the model-free algorithm used to learn the policies. Every performance curve in the graphs represents the point-wise average of at least 100 independent runs with all identical settings.

We categorize our empirical findings in three groups:

Experiment A: Initial performance boost due to generalization (regular α).
Figure 4 plots early performance obtained using the uninformative reward function (*a*) and the informative one (*b*). The step size α is 0.1, a typical value. The graphs show a performance boost due to generalization when the informative reward function is used, but no observable differences with the uninformative reward function.

Experiment B: Initial performance boost due to generalization (small α).
Figure 5 plots performance over the first 50000 episodes, a substantial allotment of learning time. The reward function used is the uninformative one, chosen to control for the apparent advantage enjoyed by broad-generalizing learners in experiment A. (As the analysis in Section 4 will show, the results observed in experiments A and B are due to different causes which this experimental setup serves to isolate.) Decreasing the step size degrades performance for any fixed setting of generalization breadth; however, the extent of this deterioration diminishes as generalization

breadth increases. Viewed differently, $\alpha = 0.5$ reveals no observed benefit to generalization. But as α decreases to 0.10 and then to 0.05, generalization becomes increasingly beneficial.

Experiment C: Eventual degradation of performance due to generalization. Experiments A and B demonstrate that generalization can improve performance while the policy is undergoing initial development or early refinement. Figure 6, on the other hand, shows that in the final count generalization proves detrimental. Figure 6 was obtained using the uninformative (*a*) and informative (*b*) reward functions. In the former case, the more challenging nature of the task favors the use of generalization for a longer time.

Note that the graphs in Figures 4–6 have vastly different x-axis scales. Moreover, the y-axis scales in Figures 4 and 5 are different from those in Figure 6.

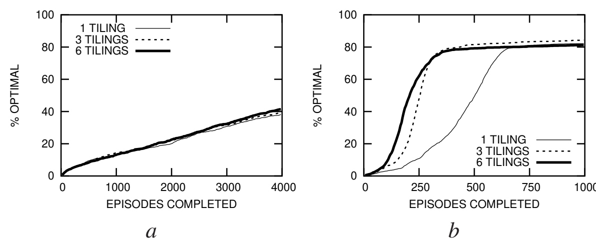


Fig. 4: Early performance with the uninformative (*a*) and informative (*b*) reward functions. The ordering of the curves in *b* is statistically significant at a 0.005 confidence level between episodes 92 and 280.

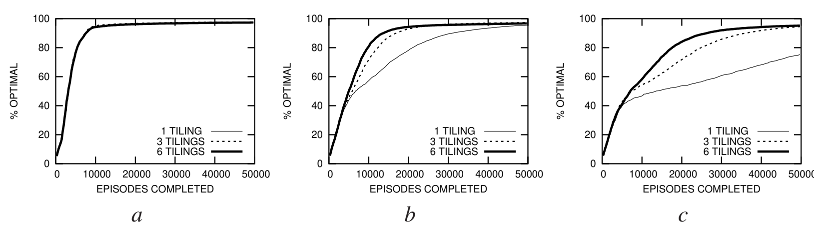


Fig. 5: Performance with the uninformative reward function and three settings of step size: $\alpha = 0.5$ (*a*), $\alpha = 0.1$ (*b*), and $\alpha = 0.05$ (*c*). The ordering of the curves is statistically significant at a 0.005 confidence level between episodes 6100 and 22100 (*b*), and 10000 and 40000 (*c*).

4 Interpretation of Empirical Results

As observed in Section 3, generalization tends to help initial performance but hurts in the long run. This section analyzes the causes of these phenomena. We start by intro-

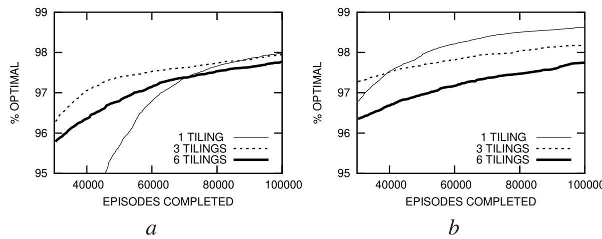


Fig. 6: Long-term performance (episodes 50000–100000) with the uninformative (*a*) and informative (*b*) reward functions. The ordering of the curves in *b* is statistically significant at a 0.005 confidence level staring at episode 55000.

ducing an abstraction of the problem. We categorize state-action pairs as *overestimated*, *underestimated*, and *correctly estimated* with respect to the backup procedure used and the approximate value function $\hat{Q}(s, a)$ for states and actions. In Q-learning, an overestimated state-action pair is characterized by

$$\hat{Q}(s, a) > r(s, a) + \gamma \max_{a' \in \mathcal{A}} \{\hat{Q}(s', a')\},$$

where s' is the successor state. Underestimated and correctly estimated state-action pairs are defined by replacing the "greater than" sign in the above equation with "less than" and "equals" signs, respectively. Note that this terminology is unrelated to the *true* values of state-action pairs under the current policy; state-action pairs are "underestimated," "overestimated," or "correctly estimated" solely with respect to *one-step* updates. Finally, we define a state-action pair (s, a) to be *desirable* if a is a near-optimal action in state s, i.e.,

$$|Q^*(s, a) - \max_{a' \in \mathcal{A}} Q^*(s, a')| < \delta,$$

where Q^* is the optimal value function and $\delta > 0$ is a small constant. *Undesirable* state-action pairs are defined symmetrically.

The effect of generalization on correctly estimated state-action pairs is nonexistent or negligible since backups in such cases generate zero expected error. The effect of generalization on overestimated and underestimated state-action pairs, on the other hand, is significant. In what follows, we analyze these two cases separately. We assume the exploration/exploitation trade-off is addressed using Boltzmann (softmax) action selection, an ϵ-greedy policy, or any other method in which the greedy action $\hat{a}^* = \arg\max_{a \in \mathcal{A}} \hat{Q}(s, a)$ is selected in state s with the greatest probability and the probabilities of selection of the other actions are nondecreasing in their value estimates. We refer to the region to which a value update of a state-action pair (s, a) is generalized as the *vicinity of* (s, a).

4.1 Generalization on overestimated vs. underestimated state-actions pairs

Generalizing the value update of an overestimated state-action pair (s, a) to nearby state-action pairs will decrease their value estimates and thus reduce the likelihood of selection of the corresponding actions in their respective states. If (s, a) and state-action pairs in its vicinity are undesirable, this generalized update is beneficial *regardless* of whether these state-action pairs are also overestimated. If, on the other hand, some state-action pairs in the vicinity of (s, a) are desirable, generalization is harmful if they are not overestimated. In this latter case, generalization will excessively lower the probability of selection of certain good actions.

Similarly, generalization on an underestimated state-action pair (s, a) is helpful if the state-action pairs in its vicinity are desirable, and may be harmful if there are undesirable pairs with correct or excessive estimates. However, there is an additional benefit to generalizing on desirable underestimated state-action pairs. Typical domains have continuous value functions. In this case, generalization ensures that the nearby state-action pairs also have favorable estimates *even if they are rarely tried*. Generalization thus accelerates the adoption of better actions in the vicinity of (s, a) as greedy choices, which is increasingly helpful with small step sizes. By contrast, a non-generalizing learner will require more exploratory visits to the vicinity of (s, a) to build up these actions' value estimates.

4.2 Application to the empirical results

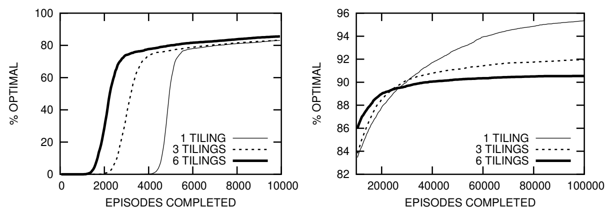
Generalization improves early performance in experiment A when used with the more informative reward function because the algorithm can more rapidly learn clusters of actions that lead to a fall off the cliff. When such a catastrophic event occurs and a heavy penalty is received (-100), the learner generalizes the outcome to neighboring p values, thus requiring less time to identify directions of travel to avoid *for any value of* p. A non-generalizing learner, on the other hand, needs to visit every p value within resolution to rule out a poor choice of direction. This is an example of the beneficial effects of generalization on overestimated state-action pairs. The uninformative reward function, on the other hand, does not communicate the undesirability of falling off the cliff and leads to no performance improvement with generalization.

The small step sizes in experiment B require a substantial amount of exploratory activity to build up value estimates for better p choices in the vicinity of an already established one—unless generalization across tiles is used, yielding elevated value estimates for those p choices even if they are rarely tried. As a result, generalization improves performance for small step sizes, a benefit of generalization on underestimated state-action pairs. Underestimated state-action pairs are common in experiment B as positive reward propagates from the faraway goal state (the only source of nonzero reward) to the rest of the grid world, one state at a time. Thus, the use of the uninformative reward function ensures that the performance differences are not due to the expedited mastery of cliff avoidance with generalization, as in experiment A.

Once the value estimates become sufficiently accurate (with the optimum actions adopted as greedy choices and the catastrophic actions assigned low values), generalization cannot further improve performance. The negative effects of generalization are, on the other hand, still at work. The empirical results in experiment C confirm that generalization is detrimental at the final stages. As expected, the observed performance degradation is monotonic in the breadth of generalization.

5 Adaptive Generalization

We have confirmed that our empirical findings in Section 3 scale with map size. As an example, Figure 7 shows the early and late performance curves using the informative reward function and a 32×8 grid world. This new map is 6.4 times larger than that of Figure 2 but structurally similar to it.



a *b*

Fig. 7: Performance in a 32×8 grid world: episodes 0–10000 (*a*) and 10000–100000 (*b*). The ordering of the curves is statistically significant at a 0.005 confidence level between episodes 2000 and 5600 (*a*) and starting at episode 40000 (*b*).

The empirical results indicate that broad generalization is helpful at the early stages of learning but detrimental in the final count, suggesting that *online adjustment* of generalization breadth would yield the optimal approach. To this end, we implemented an adaptive algorithm as follows. For every state-action pair (s, a), the method maintains a *reliability index* $\rho(s, a)$ that expresses the learner's confidence in $\hat{Q}(s, a)$, ranging from 0 (unreliable) to 1 (reliable). The reliability indices (initialized to 0) are stored in a tiling organization with the same resolution as the organization for the Q-values themselves. Backups of $\hat{Q}(s, a)$ that yield a large error *lower* the reliability indices for (s, a) and nearby state-action pairs; backups that result in a small error *increase* those reliability indices. When performing a backup, the algorithm selects the largest allowable breadth of generalization such that the state-action space covered has an average reliability index of less that $1/2$. This heuristic encourages broad generalization when the value estimates are rapidly changing and discourages generalization when they are near convergence. Note that no actual conversion from one tiling organization to another is necessary when changing generalization breadth: with an appropriate update scheme, a single flat tiling organization can efficiently *simulate* any number of tilings.

In this framework, one needs to specify only the *range* of minimum and maximum generalization breadth to be used, leaving the parameter adjustment to the algorithm. Observe that the adaptive-generalization method varies generalization as needed based not only on the learning stage (*time-variant* generalization), but also on the state-space region (*space-variant* generalization). This facility is valuable because some regions of the state-action space are visited very frequently and favor an early cutback on generalization; other state-action regions are visited only occasionally and would benefit from generalization for a longer time.

To complete the description of the adaptive-generalization algorithm, it remains to specify how a backup error of a certain magnitude affects the reliability index of the corresponding state-action pair. Various update schemes can be proposed here. Our approach increases the reliability by $1/2$ on zero error and decreases it by $1/2$ on a very large error (50 was an appropriate setting in our domain); the intermediate cases are linear interpolations between these extremes. We generalize each reliability update to its immediate vicinity. In stochastic environments, it may be additionally useful to *decay* the reliabilities periodically. Figure 8 presents the finalized adaptive-generalization algorithm in pseudocode, embedded in Q-learning.

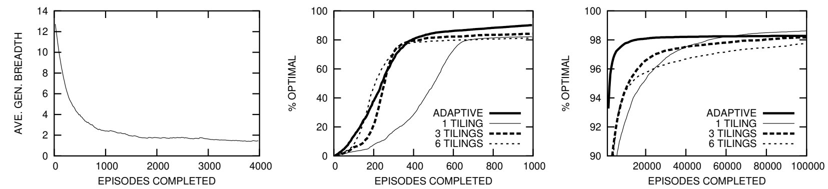
ADAPTIVE-GENERALIZATION($\mathit{max\text{-}error}$)

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
$\rho(s, a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
repeat $s \leftarrow$ current state
    $a \leftarrow \pi(s)$
    Take action a, observe reward r, new state s'
    $\mathit{error} \leftarrow [r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')] - Q(s, a)$
    $\mathit{num\text{-}tilings} \leftarrow \max\{t \geq 1 :$ avg $\rho(s'', a'')$ at most 1/2$\}$
    Update $Q(s, a)$ by $\alpha \cdot \mathit{error}$ using generalization breadth $\mathit{num\text{-}tilings}$
    $\rho(s, a) \leftarrow \left[\rho(s, a) + \frac{1}{2} - \frac{|\,\mathit{error}\,|}{\mathit{max\text{-}error}}\right]_0^1$
    $\pi \leftarrow \epsilon$-greedy w.r.t. Q
until converged

Fig. 8: Adaptive generalization method in pseudocode. The left arrow "$\leftarrow$" denotes assignment; $[x]_b^a = \max\{\min\{x, a\}, b\}$ denotes the bounding operation.

We did not attempt to optimize the above reliability-update rule and used it as an informed first guess. Figure 9*a* illustrates the progress of generalization breadth on a typical run in this scheme. Figures 9*b* and 9*c* demonstrate that even this "first guess" approach to varying generalization is superior to fixed settings of generalization breadth between episode numbers 450–49000, which arguably covers any reasonable allotment of training time in this domain.

To see why the 1-tiling (no generalization) learner eventually overtakes the adaptive learner, observe that in online RL the learner typically discovers the optimal policy much sooner than its *exact* value function. Indeed, to obtain the optimal policy the learner need only get the relative values of the states right; the actual estimates can be arbitrarily far from the true values. Even after a near-optimal policy has been discovered, the adaptive learner thus continues to see a steady drift of the values as positive reward propagates from the goal state to the rest of the grid world, one state at a time. Faced with this continual change, the above adaptive rule is too slow to cut generalization. This minor drift is easy to detect and correct for with a more informed reliability-update rule. At the same time, even our relatively simple update rule results in good performance. We conclude that even unsophisticated schemes for varying generalization breadth are generally preferable to any fixed setting.



a *b* *c*

Fig. 9: Adaptive method in the 10×4 grid world: per-episode average generalization breadth, smoothed (*a*); and comparative performance, episodes 0–1000 (*a*) and 1000–100000 (*b*). At a 0.005 confidence level, the adaptive method is superior between episodes 450 and 49000.

6 Conclusions

This paper explores parameterization issues in tile coding, a widely adopted function-approximation method for reinforcement learning. In particular, we present a precise empirical study of the effect of *generalization breadth* on the performance of a tile-coding approximator. Our findings demonstrate that generalization helps at the early stages of learning but invariably hurts past a certain point. As a result, *no single setting* achieves the best performance throughout the learning curve. We pinpoint the causes of this phenomenon and build on our analysis to propose a novel technique for *automatically adjusting* generalization breadth as needed in different regions of the state-action space (*space-variant* generalization) and at different learning stages (*time-variant* generalization). We experimentally show the superiority of varying the generalization breadth in this way to any fixed parameterization. Our adaptive-generalization method is generic and can be advantageously applied in any setting in which tile coding is used.

7 Acknowledgments

This research was supported in part by NSF CAREER award IIS-0237699 and an MCD fellowship. The authors are thankful to Lilyana Mihalkova for her feedback on an earlier version of this manuscript.

Appendix: Proof of Theorem 1

Proof. (Sketch.) The theorem can be proven by establishing that any function representable with a t-tiling organization is also representable with a single-tiling organization, and vice versa. The former claim is shown by projecting the t-tiling organization onto the single-tiling organization and weighting the tiles of the single-tiling organization by the sum of the corresponding tile weights of the t-tiling organization. The latter claim is shown by assigning random weights to the leftmost $t - 1$ tiles of the t-tiling organization (one in each of the first $t - 1$ tilings) and weighting the leftmost tile in the remaining tiling such that the sum of the t tile weights equals the weight of the first tile in the single-tiling organization; the latter weighting operation is repeated iteratively, moving at each step one tile to the right in the t-tiling organization.

References

1. Santamaria, J.C., Sutton, R.S., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behavior **6** (1997) 163–217
2. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
3. Lin, C.S., Kim, H.: CMAC-based adaptive critic self-learning control. In: IEEE Trans. Neural Networks. Volume 2. (1991) 530–533
4. Stone, P., Sutton, R.S.: Scaling reinforcement learning toward RoboCup soccer. In: Proc. 18th International Conference on Machine Learning (ICML-01), Morgan Kaufmann, San Francisco, CA (2001) 537–544
5. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Tesauro, G., Touretzky, D., Leen, T., eds.: Advances in Neural Information Processing Systems 8, Cambridge, MA, MIT Press (1996) 1038–1044
6. Tham, C.K.: Modular On-line Function Approximation for Scaling up Reinforcement Learning. PhD thesis, Cambridge University, Cambridge, England (1994)
7. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, Cambridge University (1989)