

Task Space Tile Coding: In-Task and Cross-Task Generalization in Reinforcement Learning

Lutz Frommberger

University of Bremen
AG Cognitive Systems
Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
`lutz@informatik.uni-bremen.de`

Abstract. Exploiting the structure of a domain is an important prerequisite for being able to efficiently use reinforcement learning in larger state spaces. In this paper, we show how to benefit from the explicit representation of structural features in so-called structure space aspectualizable state spaces. We introduce task space tile coding as a mechanism to achieve generalization over states with identical structural properties. This leads to a significant improvement of learning performance. Policies learned with task space tile coding can also be applied to unknown environments sharing the same structure space and thus enable for a faster learning in new tasks.

1 Introduction

Generalization is an important concept in autonomous learning task. It enables to draw conclusions from previously investigated circumstances to new situations. When the problems to learn become complex and the state space becomes larger, the ability to generalize is a crucial factor.

The insight that exploiting the structure of a domain is key to successful RL in large state spaces has already been mentioned by Thrun and Schwartz [10]. The structure of the domain are recurring properties within the state space that are significant for action selection [3]. In many scenarios we know about such structures that affect action selection in a way that the optimal action remains more or less consistent over different states. For example, in a robot navigation task, the optimal action within a corridor will usually be “go straight”. In other words, structurally similar states are expected to trigger similar actions.

A so-called *structure-space aspectualizable* state space is a state space that includes structural features into the state space representation in an easily accessible way. It has been shown that policies learned in this kind of state spaces can easily be transferred to new learning tasks that share the same structure representation [3]. In this paper, we present *task space tile coding* (TSTC) as a new method to exploit structure space aspectualizable state spaces based on the successful *tile coding* mechanism [1, 8]. TSTC is not only capable of enabling transfer of learned policies to new learning tasks, but it also allows for structural generalization *within* the original learning task which leads to rapid improvement

of learning performance. In other words, task space tile coding allows for both *in-task* and *cross-task* generalization.

2 Related Work

In RL, generalization naturally comes with the use of value function approximation (VFA). Several approaches exist to adapt VFA to the structure of the underlying domain. Many approaches rely on adaptive partitions of the state space [5, 6, 11, e.g.]. Whiteson and Stone presented an adaptive method for approximation with neural networks [13]. Mahadevan uses a global structural analysis of the state space based on Laplacian eigenfunctions to automatically construct a basis for the function approximator [4]. Tile coding is frequently used for VFA in RL [8, 7, e.g.]. There also exist adaptive approaches [14]. The main difference of all these approaches to the method proposed in this paper is that they only generalize locally and use the structure to achieve a good approximation, while task space tile coding explicitly generalizes over structural features. Tile coding also has been applied in transfer learning tasks by copying the weights of a CMAC to a new learning task [9]. This is not necessary with TSTC, where a policy can be transferred ad-hoc, without any effort.

3 Structure Space Aspectualization

We suppose the problem to solve to be represented as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ with a state space \mathcal{S} , a set of actions \mathcal{A} , a function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ denoting a probability distribution that executing action a at state s results in state s' , and a function $R : \mathcal{S} \rightarrow \mathbb{R}$ assigning a reward to any $s \in \mathcal{S}$. In the following, we shortly introduce structure space aspectualizable state spaces, as defined in [3].

3.1 Structure Space and Task Space

In a structure space aspectualizable state space, structural properties of the domain are explicitly represented as a subset of the overall state space representation, the feature vector $s = (s_1, s_2, \dots, s_n)$. That is, part of the features are exclusively a description of the state space’s structure. The feature space generated by these features is called *structure space* (\mathcal{S}_S). A function $\text{ssd} : \mathcal{S} \rightarrow \mathcal{S}_S$ returns the features of structure space. The space generated by the remaining features of s is called *task space* (\mathcal{S}_T); these features are returned by a function $\text{tsd} : \mathcal{S} \rightarrow \mathcal{S}_T$. All in all, \mathcal{S} is assumed to be represented as a Cartesian product of task and structure space representation $\mathcal{S}_T \times \mathcal{S}_S$ such that we have a state representation $s = (\text{tsd}(s), \text{ssd}(s))$.

Structure space represents the part of the problem that calls for action selection based on general properties of the domain, while task space is the part that is tightly coupled to the concrete problem to solve. Thus, action selection that is

based on structure space features only can be expected to implement a *generally sensible behavior* of an agent that performs in a reasonable way in any scenario sharing the same structure. A *structure space policy* $Q_S : \mathcal{S}_S \rightarrow \mathcal{A}$ operates exclusively on structure space. It has been shown that a structure space policy can be utilized for transfer learning, that is, it can help speeding up learning in new and unknown learning tasks [3].

3.2 In-task Generalization Based on Structure Space

The approach in [3] shows how an explicit representation of structure can be used to improve learning performance in a new learning task. However, structural similarities can also be found at different positions within the *same* state space. The goal is now to exploit structure space in a way that learned knowledge directly generalizes over states with the same structure. The idea behind generalization is that states that are—under some metric—*near* to each other require same or similar actions. We will now show how to apply this idea to structure space: If a learning agent is to be able to reuse previously gathered knowledge at a state s , it should be able to consider the information stored for a state s' with the same structure, that is, $\text{ssd}(s) = \text{ssd}(s')$.

In RL, knowledge about states is stored in the value function V (or Q-function Q , respectively). What we aim at is that whenever the value function $V(s)$ is updated, not only $V(s)$ is updated, but—preferably—*all* $V(s')$ with $\text{ssd}(s) = \text{ssd}(s')$ are updated as well, including states that have not been visited yet, and including states of which, at the time of learning, we do not know if they exist at all. If an update of a state also affects all states with the same structure representation $\text{ssd}(s)$ (and no states with a different structure representation), we call this *task space generalization*.

4 Task Space Tile Coding

In this section, we introduce task space tile coding (TSTC) as a recipe how to design the tilings and to distribute value updates according to the structural representation of the environment to achieve task space generalization.

4.1 Storing a Value Function with Tile Coding

In tile coding, we have several different partitions of the state space, called *tilings*. A tiling is a tessellation of non-overlapping grid cells; in other words, every state s belongs to exactly one cell within each tiling. Each of these cells is called a *tile*. The tilings are organized in a way that each is arranged with a certain offset in each dimension to the others such that a state is mapped to different tiles in different tilings.

Let n be the number of tilings and χ_i be one of several functions that assign each $s \in \mathcal{S}$ a representative for a tile within a certain tiling i . A hash function $h : \mathcal{S} \rightarrow \mathbb{N}$ now delivers a cell number $h(\chi_i(s))$, and $c : \mathbb{N} \rightarrow \mathbb{R}$ is a function

that returns the contents of this cell. A value $V(s)$ is equally distributed over the tilings by

$$c(h(\chi_i(s))) = V(s), \quad i = 1, \dots, n. \quad (1)$$

Conversely, a value $V(s)$ is retrieved as an average over all tilings:

$$V(s) = \frac{1}{n} \sum_{i=1}^n c(h(\chi_i(s))). \quad (2)$$

The value function can be updated via dynamic programming. For TD(0) learning, the update rule would be (with a step size α and a TD error δ):

$$c(h(\chi_i(s))) = c(h(\chi_i(s))) + \alpha\delta, \quad i = 1, \dots, n. \quad (3)$$

4.2 Design of the Tiling Function

Task space tile coding does not alter the storing mechanism of tile coding, it only affects the tiling function χ to ensure an appropriate distribution of value function updates over structurally identical states. In order to easily demonstrate how to design χ for TSTC, we make some simplifying (but not limiting) assumptions on the design of the state space. Let us have a structure space aspectualizable state space $\mathcal{S} = \mathcal{S}_T \times \mathcal{S}_S$ with both \mathcal{S}_T and \mathcal{S}_S being finite and discrete.¹ W.l.o.g. we assume that each $p \in \mathcal{S}_T$ has the form $p = (p_1, \dots, p_k)$ with $p_i \in \mathbb{N}$. For every feature dimension i exists a p_i^{\max} with $p_i^{\max} \geq p_i$. For the sake of simplicity of notation, we assume the minimal value to be 1. A vector $p^{\max} = (p_1^{\max}, \dots, p_n^{\max})$ gathers the maxima for each dimension.

For structure space \mathcal{S}_S it is also assumed w.l.o.g. that each $q \in \mathcal{S}_S$ has the form $q = (q_1, \dots, q_m)$. For each dimension there is a minimal distance between the possible features. From that emerges a distance vector $d = (d_1, \dots, d_m)$ with

$$d_i = \min_{q, \bar{q} \in \mathcal{S}, q \neq \bar{q}} (|q_i - \bar{q}_i|). \quad (4)$$

Let us now define a tiling function χ as follows:

$$\chi(s) = \chi(p_1, \dots, p_k, q_1, \dots, q_m) \quad (5)$$

$$= (p'_1, \dots, p'_k, q'_1, \dots, q'_m) \quad (6)$$

$$= \left(\frac{p_1}{p_1^{\max}}, \dots, \frac{p_k}{p_k^{\max}}, \frac{q_1}{d_1}, \dots, \frac{q_m}{d_m} \right). \quad (7)$$

Obviously, it holds that $p'_i = \frac{p_i}{p_i^{\max}} \in (0, 1]$. Furthermore, for two different structure space representations q_i and q_j ($i \neq j$) it holds that $|q'_i - q'_j| = \left| \frac{q_i}{d_i} - \frac{q_j}{d_j} \right| \geq 1$, that is, the distance between different features in structure space is always at least 1. We now choose a hypercube with an edge length of 1 in each dimension

¹ Task space tile coding is not limited to discrete state spaces, though. It can also be applied to continuous state spaces.

for our tilings. In this hypercube, all instances of $(p'_1, \dots, p'_k, q'_1, \dots, q'_m)$ with fixed q'_1, \dots, q'_m map to the same hypercube cell; and for every two non-identical q'_1, \dots, q'_m it holds that they are always mapped to different cells. To give an example: Let $s = (p_1, p_2, q_1, q_2)$ and $p_1 \in \{1, 2\}$, $p_2 \in \{1, 2, 3, 4\}$, $q_1 \in \{1, 2\}$, and $q_2 \in \{1, 3, 6\}$. Then, $p^{\max} = (2, 4)$ and $d = (1, 2)$. So $\chi(2, 2, 1, 6) = (1, 0.5, 1, 3)$, $\chi(2, 2, 2, 6) = (1, 0.5, 2, 3)$, and $\chi(1, 2, 1, 6) = (0.5, 0.5, 1, 3)$. It can be seen that any p_i stays in $(0, 1]$ and, thus, within one hypercube cell, and the q_i are at least one in distance. As an effect, no generalization occurs for structurally different states, but it does for all structurally identical states.

While tile coding usually aims at reducing the state space size, the primary goal of TSTC is to exploit the generalization capabilities of tile coding. To avoid a loss of resolution in a discrete state space, we must ensure to have a sufficiently large resolution to store individual values $V(s)$ for any $s \in \mathcal{S}$. This can be achieved by choosing a sufficient number of tilings. A number of $n = \max_i p_i^{\max}$ tilings can provide this requirement and conserves the initial resolution of the discrete state space, so that each $s \in \mathcal{S}$ can still be distinguished between within the value function representation. Thus, in a finite and discrete state space, the stored value function V can converge to the optimal value function V^* .

With a tiling function as defined above we observe the following effects:

1. Generalization is exclusively applied to task space \mathcal{S}_T ; structure space \mathcal{S}_S is not matter to generalization.
2. An update of a value affects *all* states with the same structure space representation. The impact on every single state depends on the similarity of the task space representation $\text{tsd}(s)$.
3. Updates affect every observation, even those that have not been perceived and updated before.

So whenever we encounter a state s with an already known structure space representation $\text{tsd}(s)$, there is already structural knowledge provided by the stored value function which can be directly utilized within the running task. This is especially beneficial in large state spaces, where a large number of new state samples are discovered during the learning process.

4.3 Empirical Evaluation: a Grid Example

First, we demonstrate the effectiveness of TSTC in a toy example: a goal-directed navigation task in a discrete grid world that resembles the well-know *puddle world* scenario [8]. The MDP we solve consists of $N \times M$ cells where a navigating agent is supposed to travel to a distinguished goal position $(N - 1, M - 1)$ from any location within the world by applying four actions to move deterministically one cell up, down, left, or right (moving outside the grid results in no change of position). At the goal state, it receives a reward of 100. A subset of cells are *puddles* that return a negative reward of -20 . Furthermore, each movement yields a reward of -2 . The puddles are arranged in a way such that they can be regarded as structural information: Each border cell is a puddle, and additional

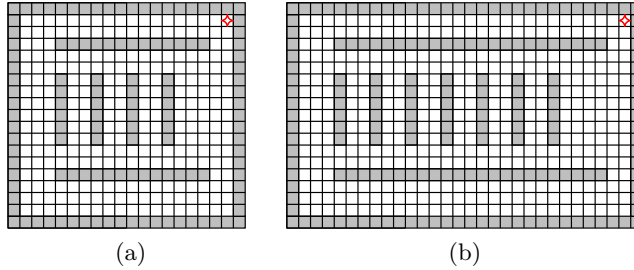


Fig. 1: Grid worlds used for evaluation. The gray cells are puddle cells, the goal location in the upper right corner is marked.

puddle formations are distributed within the world. Figure 1 shows two examples. The original state space \mathcal{S} is $(x, y) \in \mathbb{N}^2$ with $1 \leq x \leq N$ and $1 \leq y \leq M$.

We now define a structure space aspectualizable state space \mathcal{S}' . For task space, we choose the coordinates (x, y) of the grid cell. For structure space, we give the agent some additional sensory capacity to “see” whether there is (1) or is not (0) a puddle up, down, left, or right of its position. So we get a vector $t \in \{0, 1\}^4$ that denotes whether action i will bring the agent to a puddle. The overall state space representation is $(x, y, t_0, t_1, t_2, t_3)$. Note that there exists a bijection between \mathcal{S} and \mathcal{S}' . Thus, in a plain RL task, it makes no difference whether we operate on \mathcal{S} or \mathcal{S}' . With TSTC, N and M are chosen for p_i^{\max} ($i \in \{0, 1\}$). We now solve the given MDP over \mathcal{S}' with Q-learning [12] with and without TSTC. In every learning episode, the agent starts at a random location; the episode ends when the agent has reached the goal or after a certain number of actions. We use a step size $\alpha = 0.1$ and ϵ -greedy exploration with $\epsilon = 0.3$.

Figure 2 summarizes the learning results in two worlds with 20×20 and 30×20 cells (as depicted in Fig. 1) averaged over 100 test trials: With TSTC, the learned policy reaches the maximum reward much earlier than without TSTC. Also, due to structural generalization, the size of the state space has little impact on the

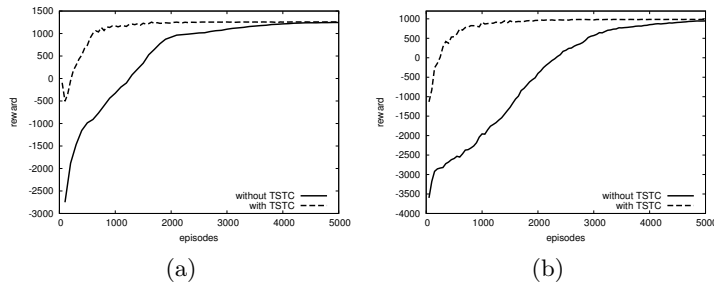


Fig. 2: Comparison of learning performance: Accumulated rewards in the puddle grid scenario with and without TSTC in the worlds in Fig. 1.

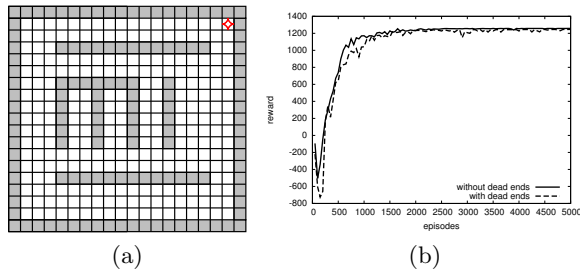


Fig. 3: Performance with TSTC with partially contradicting structural information. (Curves are scaled to the same maximum reward.)

learning performance with TSTC; with plain Q-learning, the learning speed in the larger environment is considerably slower.

TSTC is very effective in the given environments, because they are structurally homogeneous. In the worlds in Fig. 1, the optimal action is always the same for states with the same structure space representation (e.g., when observing a puddle to the left, it is always appropriate to go up). To test the TSTC performance in a world where the structural information is partially contradicting with regard to the optimal action, we repeat the learning trial in a modified environment. The world in Fig. 3a contains two “dead ends” within which, in contrast to other locations, it is *not* optimal to go up if there is a puddle to the left. Figure 3b compares the performance in the world with (Fig. 3a) and without (Fig. 2a) dead ends: in the contradicting world learning takes slightly longer and is a bit less stable than in the homogeneous world. However, the effect is very subtle, and the maximum reward is still achieved much faster than without TSTC.

4.4 Empirical Evaluation: Goal-directed Navigation

Obviously, TSTC has its strengths especially in larger state spaces. Thus, we now evaluate it in another similar, but significantly larger scenario: a simulated indoor robot navigation task. The robot has a diameter of 50 cm and is able to perceive and uniquely identify walls around it within a range of 3.5 meters. It is capable of performing three different actions: moving forward by 2.5 cm and turning by 4 degrees both to the left and to the right while also moving the agent forward for 2.5 cm. A small amount of noise is added to all actions. There is no built-in collision avoidance or any other navigational intelligence provided. See Fig. 4 for a look on the simulation testbed.

The original state space in the scenario is continuous; for efficient learning, it is discretized by using an egocentric structure space aspectualizable state space abstraction called *le-RLPR* [2], which we shortly introduce in the following.

We can encode an approximate qualitative position of the agent within the world by regarding a sequence of identified walls c_i at 7 discrete angles around

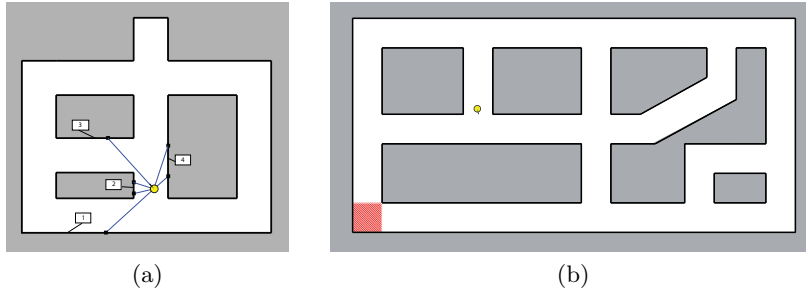


Fig. 4: The simulated indoor robot environment used for evaluation. (a) shows 4 identified walls at discrete angles: $\text{tsd}(s) = (1, 2, 2, 3, 0, 4, 4)$. The experiment was run in (b), the goal area is marked in the lower left corner.

the robot. The optimal action selection for this position changes with any new goal and any new environment, so this refers to task space. Each detected wall has a unique number starting with 1: $\text{tsd}(s) = (c_1, \dots, c_7)$. To encode structure space, we disregard the wall identifiers. Instead, we use the *relative line position representation* (RLPR). RLPR roughly encodes the relative position of line segments perceived by the agent’s sensory system relative to its moving direction. The space around the agent is partitioned into bounded and unbounded regions R_i (see Figure 5). Two functions $\bar{\tau} : \mathbb{N} \rightarrow \{0, 1\}$ and $\bar{\tau}' : \mathbb{N} \rightarrow \{0, 1\}$ are defined: $\bar{\tau}(i)$ denotes whether there is a line segment detected within a region R_i and $\bar{\tau}'(i)$ denotes whether a line spans from a neighboring region R_{i+1} to R_i . The structure space representation $\text{ssd}(s)$ is defined as $(\bar{\tau}'(R_1), \dots, \bar{\tau}'(R_6), \bar{\tau}(R_9), \dots, \bar{\tau}(R_{13}))$, and le-RLPR is the concatenation of task and structure space representation: $s = (c_1, \dots, c_7, \bar{\tau}'(R_1), \dots, \bar{\tau}'(R_6), \bar{\tau}(R_9), \dots, \bar{\tau}(R_{13}))$

To evaluate the impact of TSTC on this scenario, we used Watkins’ Q(λ) algorithm with discounted rewards for learning [12]. During training, the agent used an ϵ -greedy policy with $\epsilon = 0.2$ for exploration. It started from 1000 starting positions randomly distributed over the corridors. Every 500 episodes, the success was evaluated on a cross-validation set of 200 starting positions disjoint from the ones used for training. A step size of $\alpha = 0.005$, a discount factor of $\gamma = 0.98$, and $\lambda = 0.9$ was used with TSTC. For the non-TSTC case, we used

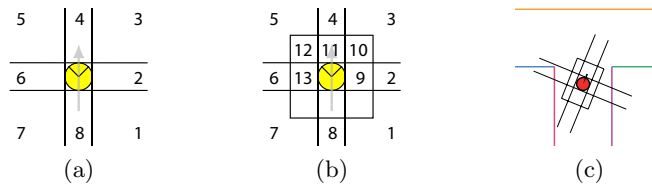


Fig. 5: The RLPR grid: Neighboring regions R_1 to R_{13} around the robot in relation to its moving direction. Walls cutting grid borders are shown in (c).

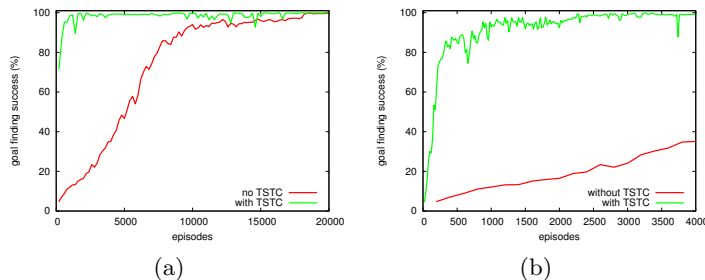


Fig. 6: Effect of task space tile coding: Compared to “ordinary” le-RLPR, the goal finding success increases rapidly with TSTC (a). In (b), we zoom into the first episodes to show them on a smaller scale.

$\alpha = 0.02$. A reward of 100 is given when a goal area is reached and -100 if the agent collides with a wall. A learning episode ends when the agent reaches the goal state, collides with a wall, or after a certain number of actions.

The effect of task space tile coding is dramatic. After a few dozen episodes it becomes visible that the agent develops a way to cope with structures in the world and begins to follow corridors and turn around curves. This leads to a thorough exploration of the environment very quickly. Thus, near 100% goal finding success is reached after 1,000 episodes instead of 10,000 episodes without TSTC (see Fig. 6 for a comparison).

5 Transfer Learning with Task Space Tile Coding

Task space tile coding generalizes over states with the same structure space representation, even if the states are previously unknown. This also holds for states in another, unknown environment, as long as it shares the same structure space. Thus, task space tile coding is not only able to *in-task*, but also *cross-task* generalization: knowledge gained in one task can be used to improve the learning performance in another task.

5.1 Ad-hoc Structure Space Policy Transfer

When confronted with a previously unvisited state, all the knowledge encoded in the value function originates from updates caused by structurally identical states. Thus, action selection in this case will only rely on structure space information $ssd(s)$. In an unknown environment, a policy learned with TSTC directly implements a structure space policy as described in Sect. 3.1. As no additional effort has to be taken to create a structure space policy, a policy learned in a source task \mathcal{S} can be immediately applied to a target task \mathcal{S}' and will show a generally sensible behavior with regard to the structural elements there. Therefore, we refer to transfer of policies learned with TSTC as *ad hoc transfer*.

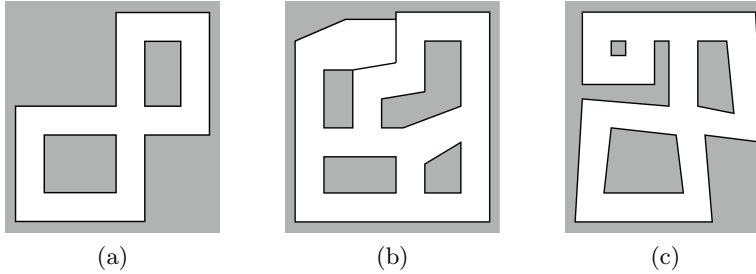


Fig. 7: Structurally similar environments used for evaluating the performance of structure space policies.

Two prerequisites have to be ensured for successful ad hoc transfer before learning the source task: First, no two task space representations in source and target space may be identical. That is, given source and target tasks with state spaces \mathcal{S} and \mathcal{S}' and the corresponding task spaces \mathcal{S}_T and \mathcal{S}_T' , it must hold that $x \in \mathcal{S}_T \Rightarrow x \notin \mathcal{S}_T'$. This can be achieved by assigning a fixed offset to coordinates or when enumerating features. Second, p^{\max} has to be chosen appropriately to make sure that generalization also affects the task space of the target task. This has to be considered *before* starting to solve the source task. Each p_i^{\max} has to be chosen greater than the maximum observable feature in both source and target tasks. As features in both tasks have to be represented mutually exclusively (see above), it must hold that $p_i^{\max} \geq \max(p_i, p'_i)$ with $p \in \mathcal{S}_T$ and $p' \in \mathcal{S}_T'$.

5.2 Evaluation: Structure Space Policies with TSTC

A policy learned with TSTC is a structure space policy and can be immediately applied to a new environment. We demonstrate this in the goal-directed robot navigation scenario described in Sect. 4.4. We used policies learned in the environment in Fig. 4 for a varying number of learning episodes (ranging from 500 to 50,000) and applied them to the unknown environments depicted in Fig. 7.

In these target worlds, $\text{tsd}(s)$ was set to a bogus value that is not existent in the source task. Then the agent started from 1,000 starting positions spread over the corridors. A test run in the destination task was regarded as successful if the robot was able to navigate within the corridors without colliding for a certain number of time steps. No learning takes place in the target worlds. The results are summarized in Table 1: In the simple world in Fig. 7a, the robot is navigating almost perfectly successfully even with policies that learned only for 1,000 episodes. In the more complex environments (Fig. 7b and 7c) which also include corridor layouts not present in the source environment, it takes longer to learn a successful structure space policy, but success rates of more than 80% can still be achieved. This shows that policies learned with TSTC can be used for transfer learning tasks: A structure space policy can be used as an initial value function in a new learning task. As a result, the agent then follows the learned

episode	World (a)	World (b)	World (c)
500	69.2	10.6	5.7
1,000	98.2	26.7	33.7
2,500	98.4	39.7	52.2
5,000	100	40.6	37.0
10,000	100	62.6	73.5
30,000	100	84.4	84.4
50,000	100	98.2	87.3

Table 1: Performance of structure space policies generated with TSTC in the environments shown in Fig. 7: percentage of collision-free test runs.

structure space policy in unknown circumstances, leading to a generally sensible exploration behavior and improved learning performance [3].

6 Discussion

Structure space aspectualizable state spaces do usually not occur by themselves, but are the result of an expert’s state space design based on his domain knowledge. However, the performance improvements TSTC offers justify this effort. Especially when the application aims at rapid adaptation based on few samples, TSTC can bring RL towards applications with online learning.

The decision which features to include to structure space must be taken with care. TSTC is likely to show bad performance if the action following from a structural description $ssd(s)$ is ambiguous over the state space. For example, if in the grid world example in Sect. 4.3 the goal would be in the middle of the environment, the optimal actions for states with $ssd(s) = (0, 0, 0, 0)$ would be equally distributed over \mathcal{A} , and no sensible generalization could take place. In this case, it would be reasonable to exclude certain structures from generalization, which can be achieved by an extension that adapts p^{\max} depending on the given structure (in this case, $p_i^{\max} = 1$ should be chosen for all i for $ssd(s) = (0, 0, 0, 0)$).

It might occur that p^{\max} cannot be determined in advance, but that is not too severe, as it will not break the general effect of task space tile coding. In the case that p_i^{\max} is chosen too small, generalization will only apply to a subset instead of to *all* structurally identical states, which will result in a lower amount of generalization and learning speed.

Finally, in this paper we defined TSTC on discrete state spaces. In general, TSTC can also be applied to continuous state spaces, but then, of course, the property of maintaining the original state space resolution becomes obsolete.

7 Summary

In this paper we introduced task space tile coding (TSTC) as a method to provide both in-task and cross-task generalization in structure space aspectualizable

state spaces. TSTC provides a strong generalization over states with the same representation of structural features and thus allows to reuse knowledge in circumstances sharing the same structure without losing the ability to assign an individual value in the value function for any state. This results in a dramatically improved learning performance especially in large state spaces. Policies learned with TSTC can also be applied to unknown environments sharing the same structure space and thus enable for a faster learning in new tasks.

Acknowledgments This work was supported by the DFG Transregional Collaborative Research Center SFB/TR 8 “Spatial Cognition”. Funding by the German Research Foundation (DFG) is gratefully acknowledged.

References

1. Albus, J.S.: Brain, Behavior, and Robotics. Byte Books, Peterborough, NH, USA (1981)
2. Frommberger, L.: Learning to behave in space: A qualitative spatial representation for robot navigation with reinforcement learning. *International Journal on Artificial Intelligence Tools* 17(3), 465–482 (Jun 2008)
3. Frommberger, L., Wolter, D.: Structural knowledge transfer by spatial abstraction for reinforcement learning agents. *Adaptive Behavior* 18(6), 507–525 (2010)
4. Mahadevan, S.: Samuel meets Amarel: Automatic value function approximation using global state space analysis. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI-05)*. pp. 877–917. Pittsburgh, PA (Jul 2005)
5. Munos, R., Moore, A.: Variable resolution discretizations for high-accuracy solutions of optimal control problems. In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence (IJCAI)*. pp. 1348–1355 (1999)
6. Reynolds, S.I.: Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*. Morgan Kaufmann, San Francisco (Jun 2000)
7. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior* 13(3), 165 (2005)
8. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse tile coding. In: *Proceedings of NIPS 1995*. pp. 1038–1044 (1996)
9. Taylor, M.E., Stone, P.: Behavior transfer for value-function-based reinforcement learning. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 53–59 (2005)
10. Thrun, S., Schwartz, A.: Finding structure in reinforcement learning. In: *Proceedings of NIPS 1994* (1995)
11. Uther, W.T.B., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI-98)*. pp. 769–775. Madison, WI (1998)
12. Watkins, C.: Learning from Delayed Rewards. Ph.D. thesis, Cambridge University (1989)
13. Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research* 7 (May 2006)
14. Whiteson, S., Taylor, M.E., Stone, P.: Adaptive tile coding for value function approximation. Tech. Rep. AI-TR-07-339, University of Texas at Austin (2007)