Complexity Analysis of Real-Time Reinforcement Learning Applied to Finding Shortest Paths in Deterministic Domains

Sven Koenig and Reid G. Simmons

December 1992

CMU-CS-93-106

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was supported in part by NASA under contract NAGW-1175.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the U.S. government.

Keywords: Machine Learning (Reinforcement Learning), Learning/Adaptation (Exploration), Search (Real-Time Search), Planning (Path Planning)

Abstract

This report analyzes the complexity of on-line reinforcement learning algorithms, namely asynchronous real-time versions of Q-learning and value-iteration, applied to the problems of reaching any goal state from the given start state and finding shortest paths from all states to a goal state. Previous work had concluded that, in many cases, initially uninformed (i.e. tabula rasa) reinforcement learning was exponential for such problems, or that it was tractable (i.e. of polynomial time-complexity) only if the learning algorithm was augmented. We prove that, to the contrary, the algorithms are tractable with only a simple change in the task representation ("penalizing the agent for action executions") or initialization ("initializing high"). We provide tight bounds on their worst-case complexity, and show how the complexity is even smaller if the state space has certain special properties. We compare these reinforcement learning algorithms to other uninformed on-line search methods and to informed off-line search methods, and investigate how initial knowledge of the topology of the state space can decrease their complexity. We also present two novel algorithms, the *bi-directional Q-learning algorithm* and the *bi-directional value-iteration algorithm*, for finding shortest paths from all states to a goal state, and show that they are no more complex than their counterparts for reaching a goal state from a given start state. The worst-case analysis of the reinforcement learning algorithms is complemented by an empirical study of their average-case complexity in three domains.

1 Introduction

Consider the problem for an agent of finding its way to one of a set of goal locations, where actions consist of moving from one intersection (state) to another (on a map, on a directed graph, or in a state space, see Figure 1). Initially, the agent has no knowledge of the topology of the state space. We consider two different tasks: reaching any goal state and finding shortest paths from every state to a goal state. [27] call the former task **satisficing search** (search for "all-or-none solutions"), whereas the latter one corresponds to **shortest-path search** for all states.

Off-line search methods, which first derive a plan that is then executed, cannot be used to solve the path planning tasks, since the topology of the state space is initially unknown to the agent and can only be discovered by exploring: i.e. executing actions and observing their effects. Thus, the path planning tasks have to be solved on-line. **On-line search** methods, also called incremental or **real-time search** methods [17], interleave search with action execution, see Figure 2. The algorithms that we describe here perform only minimal computation between action executions, choosing only which action to execute next, and basing this decision only on information local to the current state[1] of the agent (and perhaps its immediate successor states). This way, the time between action executions is linear in the number of actions available in the current state. If this number does not depend on the size of the state space, then neither does the search time between action executions. Such methods have recently been proven to be very powerful if executed by multiple agents [12].

There is a potential trade-off between exploitation and exploration when selecting an action, see [33]. **Exploitation** means to behave optimally according to the current knowledge, whereas **exploration** means to acquire new knowledge. Exploration consumes time, but may subsequently allow the agent to solve the task faster. Exploration is necessary for the path planning tasks, since the agent initially has no knowledge of the topology of the state space. A standard strategy to deal with the exploitation exploration trade-off is to exploit most of the time and to explore only from time to time. For exploration, the agent has to overcome the **limited experimentation problem**, since it is constrained to execute only one action of its choice in the current state, which then uniquely determines the new state of the agent.[2] Furthermore, it might not be able to reverse the effect of an action immediately, i.e. to backtrack to its former state, since there might not be an action that leads from its new state back to its former state.

We will investigate a class of search algorithms which perform reinforcement learning. The application of reinforcement learning to on-line path planning problems has been

[1] When we talk about "the current state of the agent," we always refer to a state of the state space and not to the knowledge of the agent, i.e. not to the values of the variables of the algorithm that controls the agent.

[2] Other researchers, for example [30], [38], [24], and [19], have proposed planning schemes for the case that arbitrary (not just local) actions can be executed at any time (in a mental model of the state space, i.e. as part of a "Gedankenexperiment"). Our assumptions are more restrictive.

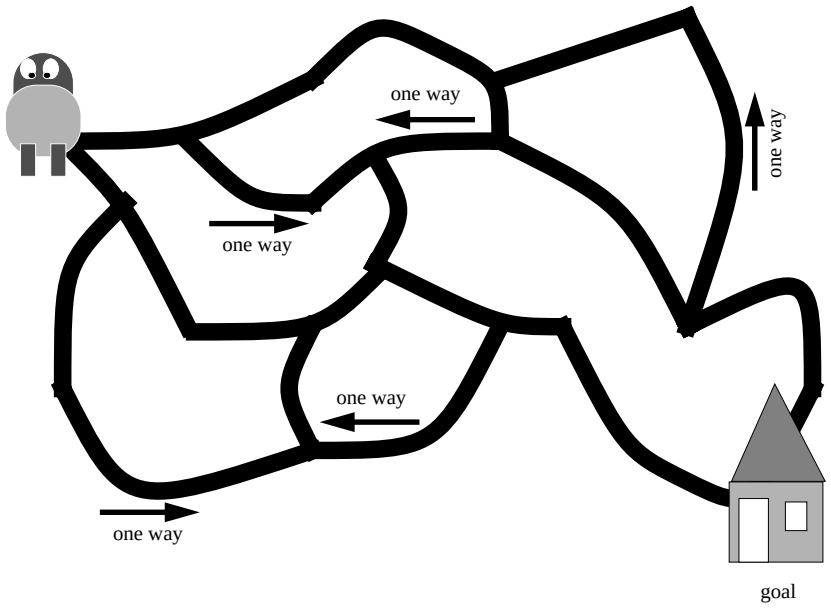

Figure 1: Navigating on a map

studied by [2], [5], [23], [19], [24], and others. [35] showed that reaching a goal state with uninformed (i.e. tabula rasa) reinforcement learning methods can require a number of action executions that is exponential in the size of the state space. [33] has shown that by augmenting reinforcement learning algorithms, the problem can be made tractable. We address the question raised by these results whether one has to augment standard reinforcement learning algorithms to make them tractable. We will show that, contrary to prior belief, reinforcement learning algorithms are tractable without any need for augmentation, i.e. their run-time is a small polynomial in the size of the state space. All that is necessary is a change in the way the state space (task) is represented.

In this report, we use the following notation.[3] S denotes the finite set of states of the state space, and G (with $\emptyset \neq G \subseteq S$) is the non-empty set of goal states. $s_{start} \in S$ is the start state of the agent. A **domain** is a state space together with a start state and a set of goal states. $A(s)$ is the finite set of actions that can be executed in $s \in S$. The size of the state space is $n := |S|$, and the total number of actions is $e := \sum_{s \in S} |A(s)|$ (i.e. an action that is applicable in more than one state counts more than once). Executing an action causes a deterministic state transition that depends only on the action and the state it is executed in (Markov assumption). $succ(s, a)$ is the uniquely determined successor state when $a \in A(s)$ is executed in $s \in S$. The **distance** $d(s, s')$ between $s \in S$ and $s' \in S$ is defined to be the (unique) solution of the following set of

[3] We use the following sets of numbers: $\mathcal{R}$ is the set of reals, $\mathcal{N}_0$ is the set of non-negative integers, $\mathcal{N}^-$ is the set of negative integers, and $\mathcal{N}_0^-$ is the set of non-positive integers.

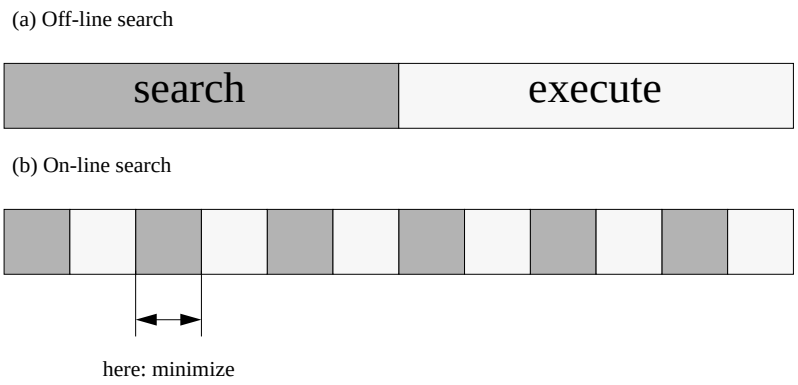

Figure 2: Off-line versus on-line search

equations

$$d(s,s') = \begin{cases} 0 & \text{if } s = s' \\ 1 + \min_{a \in A(s)} d(succ(s,a),s') & \text{otherwise} \end{cases} \qquad \text{for all } s,s' \in S$$

i.e. it is the smallest number of action executions required to reach state s' from state s. The **goal distance** $gd(s)$ of $s \in S$ is defined to be $gd(s) := \min_{s' \in G} d(s,s')$. Similarly, the **depth of the state space** d is defined to be $d := \max_{s,s' \in S} d(s,s')$, i.e. the maximum of the distances between two arbitrary states. We assume that the state space is **strongly connected** (or, synonymously, irreducible), i.e. every state can be reached from every other state (formally: $d(s,s') < \infty$ for all $s,s' \in S$). Since the state space is strongly connected, it holds that $d \leq n-1$ and $n \leq e$. We also assume that the state space is totally observable, i.e. the agent can determine its current state with certainty.[4] It also knows at every point in time which actions it can execute in its current state and whether its current state is a goal state. Furthermore, the domain is single agent and stationary, i.e. does not change over time.

Formally, the results of the report are as follows. If a good task representation ("penalizing the agent for action executions") or suitable initialization ("initializing high") is chosen, the worst-case complexity of reaching a goal state has a tight bound of $O(n^3)$ action executions for Q-learning (provided that the state space has no duplicate actions) and $O(n^2)$ action executions for value-iteration. If the agent has initial knowledge of the topology of the state space or the state space has additional properties, these bounds can be decreased further. In addition, we show that reinforcement learning methods for finding *shortest* paths from *every* state to a goal state are no more complex than reinforcement learning methods that simply reach a goal state from the start state. (All proofs can easily be derived by induction and are stated in the ap-

[4] [21] state results about the worst-case complexity of every algorithm for cases where the states are partially observable or hidden, i.e. not observable at all.

pendix.) This demonstrates that one does not need to augment reinforcement learning algorithms to make them tractable.

2 Reinforcement Learning

Reinforcement learning is learning from positive and negative rewards. (Negative rewards are often called penalties or costs). Every action[5] $a \in \overline{A}(s)$ has an immediate reward $\overline{r}(s,a) \in \mathcal{R}$, that is obtained when the agent executes the action. If the agent starts in $\overline{s}_{start} = s \in \overline{S}$ and executes actions for which it receives immediate reward $\overline{r}_t$ at step (number of actions executed previously) $t \in \mathcal{N}_0$, then the total reward that the agent receives over its lifetime for this particular behavior is

$$U(s) := \sum_{t=0}^{\infty} \gamma^t \overline{r}_t \tag{1}$$

where $\gamma \in (0,1]$ is called the discount factor. We say that discounting is used if $\gamma < 1$, otherwise no discounting is used.

Reinforcement learning algorithms find a behavior for the agent that maximizes the total reward for every possible start state.[6] Such a behavior is usually specified as a **stationary, deterministic policy**, in the following called policy. A policy, also called action map or state-action rules,

$$f : \overline{S} \rightarrow \bigcup_{s \in \overline{S}} \overline{A}(s)$$

where $f(s) \in \overline{A}(s)$ for all $s \in \overline{S}$, is an assignment of actions to states that determines which action $f(s) \in \overline{A}(s)$ the agent has to execute in its current state $s \in \overline{S}$. Executing a policy makes the agent highly reactive. By using closed loop plans instead of open loop plans, the agent is able to deal with uncertain action outcomes ("contingencies") and thus overcomes some of the deficiencies caused by more traditional planning approaches. Although the notion "policy" originated in the field of Stochastic Dynamic Programming, similar schemes have been proposed in the context of Artificial Intelligence, for example Schopper's universal plans [26].

Finding an optimal policy solves the **credit-assignment problem**, i.e. which action(s) of an action sequence to blame if the total reward is non-optimal. Usually it is not sufficient to always execute the action with the largest immediate reward, because executing actions with small immediate rewards can be necessary to make large future rewards possible. This is called the **problem of delayed rewards** or, alternatively, reinforcement learning with delayed rewards.

[5] To describe reinforcement learning, we use the symbols of the path planning domain, but overline them.

[6] Depending on the reinforcement learning problem, the total reward can become (plus or minus) infinity if no discounting is used, and might then not be useful to discriminate between good and bad behavior of the agent. In this case, one can use either the total *discounted* reward or the average reward per step in the limit.

Reinforcement learning algorithms proceed in two steps to find an optimal policy:

1. First, they determine the values $U^{opt}(s)$ for all $s \in \overline{S}$. $U^{opt}(s)$ is the largest total reward possible for $\overline{s}_{start} = s$. These values are the solutions of the following set of equations

$$U^{opt}(s) = \max_{a \in \overline{A}(s)} (\overline{r}(s,a) + \gamma U^{opt}(\overline{succ}(s,a))) \qquad \text{for all } s \in \overline{S} \tag{2}$$

   If $\gamma < 1$, then the solution is unique. The solution might not be unique for $\gamma = 1$, but the differences $U^{opt}(s) - U^{opt}(s')$ for all $s, s' \in \overline{S}$ are [8]. Thus, any solution determines the same preferences among the states.

2. Then, an optimal policy is to execute action $\text{argmax}_{a \in \overline{A}(s)}(\overline{r}(s,a) + \gamma U^{opt}(\overline{succ}(s,a)))$ in state $s \in \overline{S}$.[7]

We analyze two reinforcement learning algorithms that are widely used: Q-learning [34] and value-iteration [4]. Both can be used off-line, and especially value-iteration has traditionally been used by updating all states either synchronously, for example simultaneously or by performing a sweep over the state space and updating every state in turn ("Gauss-Seidel updating"), or asynchronously. Since both Q-learning and value-iteration are temporal difference methods and therefore incremental, one can interleave them with action execution to construct asynchronous real-time forms that use actual state transitions. In the following, we investigate these on-line versions: 1-step Q-learning and 1-step value-iteration.

2.1 Q-Learning

The 1-step **Q-learning algorithm**[8] [35], in the following called Q-learning, consists of a termination checking step (line 2), an action selection step (line 3), an action execution step (line 4), and a value update step (line 5), see Figure 3. For now, we leave the initial Q-values unspecified.

The action selection step implements the exploration rule ("which state to go to next"). It is constrained to look only at information local to the current state s of the agent. Q-learning does not learn or use an action model, i.e. the action selection step does not need to predict $\overline{succ}(s,a)$ for an $a \in \overline{A}(s)$. Instead, information about the relative goodness of the actions is stored in the states. This includes a value $Q(s,a)$ in state s for each action $a \in \overline{A}(s)$. $Q(s,a)$ approximates the optimal total reward received if the agent starts in s, executes a, and then behaves optimally.

[7] read: "any action $a \in \overline{A}(s)$ for which $\overline{r}(s,a) + \gamma U^{opt}(\overline{succ}(s,a)) = \max_{a' \in \overline{A}(s)}(\overline{r}(s,a') + \gamma U^{opt}(\overline{succ}(s,a')))$". If several actions tie, an arbitrary one of the equally good actions can be selected.

[8] Since we consider only actions with deterministic outcomes, we state the Q-learning algorithm with the learning rate α set to one.

1. Set $s :=$ the current state.
2. If $s \in \overline{G}$, then stop.
3. Select an action $a \in \overline{A}(s)$.
4. Execute action a.
   /* As a consequence, the agent receives reward $\overline{r}(s,a)$ and is in state $\overline{succ}(s,a)$. Increment the number of steps taken, i.e. set $t := t+1$. */
5. Set $Q(s,a) := \overline{r}(s,a) + \gamma U(\overline{succ}(s,a))$.
6. Go to 1.

where
$U(s) := \max_{a \in \overline{A}(s)} Q(s,a)$
at every point in time.

Figure 3: The Q-learning algorithm

The action selection step can use the Q-values, but does not need to. The actual selection strategy is left open: It could, for example, select an action randomly, select the action that it has executed the least number of times, or select the action with the largest Q-value. Exploration is termed **undirected** [33] if it uses only the Q-values (or no information at all), otherwise it is termed **directed**. The basic Q-learning algorithms studied by us do not maintain other local information than the Q-values and are therefore undirected.

Once the selected action a has been executed in state s and the immediate reward $\overline{r}(s,a)$ has been received, the agent temporarily has access to the Q-values of its former and its new state at the same time, and the value update step adjusts $Q(s,a)$. The 1-step look-ahead value $\overline{r}(s,a) + \gamma U(\overline{succ}(s,a))$ is more accurate than, and therefore replaces, $Q(s,a)$. The value update step can also adjust other information local to the former state if needed.

The Q-learning algorithm is memoryless. We call an on-line search algorithm **memoryless** [14] if it cannot remember information other than what it has stored in the states. Its access to this information is restricted to information local to the current state of the agent. After the execution of an action, the algorithm can only propagate information from the new state of the agent to its former state. Since the algorithm cannot propagate information in the other direction and has no internal memory, it cannot easily accumulate information: Information that it had available in its former state is inaccessible in its new state. The notion "memoryless" is motivated by the observation that the size required for the internal memory of an on-line search algorithm should not depend on the size of the state space. If one actually builds a robot, then one can give it only an internal memory of finite size. If this robot is to solve tasks in state spaces of arbitrary size, then the size of the internal memory needed by the algorithm that controls the robot cannot depend on n. Thus, only algorithms that

1. $s :=$ the current state.
2. If $s \in \overline{G}$, then stop.
3. Select an action $a \in \overline{A}(s)$.
4. Execute action a.
   /* As a consequence, the agent receives reward $\overline{r}(s,a)$ and is in state $\overline{succ}(s,a)$. Increment the number of steps taken, i.e. set $t := t+1$. */
5. Set $U(s) := \max_{a \in \overline{A}(s)}(\overline{r}(s,a) + \gamma U(\overline{succ}(s,a)))$.
6. Go to 1.

Figure 4: The value-iteration algorithm

merely need a memory of constant size (i.e. finite state machines) can work unchanged in arbitrarily large state spaces. A memoryless algorithm is an extreme example of such algorithms.

2.2 Value-Iteration

The 1-step **value-iteration algorithm**, in the following called value-iteration, is similar to the 1-step Q-learning algorithm, see Figure 4. Like the Q-learning algorithm, it is memoryless. The difference is that the value-iteration algorithm can access $\overline{r}(s,a)$, $U(\overline{succ}(s,a))$ and other information of state $\overline{succ}(s,a)$ for every action $a \in \overline{A}(s)$ in the current state s, whereas the Q-learning algorithm has to estimate them with the Q-values (and other information local to the current state). This difference is illustrated graphically in Figure 5. Thus, the action selection step of the value-iteration algorithm (line 3) can always use the current value of $U(\overline{succ}(s,a))$ to evaluate the goodness of executing a in s, even if this value has changed since the last execution of a in s. The expectation $Q(s,a)$, however, changes only when a is executed in s and can therefore be out-dated. The value update step (line 5) becomes "Set $U(s) := \max_{a \in \overline{A}(s)}(\overline{r}(s,a) + \gamma U(\overline{succ}(s,a)))$" for value-iteration.

Value-iteration shares with Q-learning that it does not explicitly infer the topology of the state space (i.e. it does not know or learn a map), but it must know an **action model** (i.e. be able to predict $\overline{succ}(s,a)$ for all $a \in \overline{A}(s)$ in the current state $s \in \overline{S}$).[9] Whereas Q-learning does not know the effect of an action before it has executed it at least once, value-iteration only needs to enter a state at least once to discover all of its

[9] The agent knows a **map** if it is able to predict $\overline{succ}(s,a)$ for all $s \in \overline{S}$ and $a \in \overline{A}(s)$, no matter which state it is in. The agent knows an **action model** (a "distributed map") if it is able to predict $\overline{succ}(s,a)$ for all $a \in \overline{A}(s)$ in its current state $s \in \overline{S}$. It is not necessarily able to predict the outcomes of actions that are executed in other states than its current state. Thus, knowing a map is more powerful than knowing an action model.

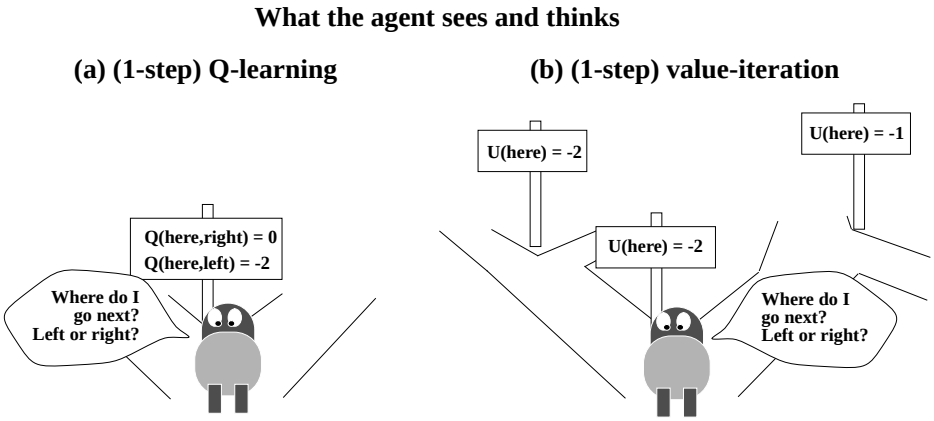

Figure 5: Reinforcement learning algorithms

successor states. Since value-iteration is more powerful than Q-learning, we expect it to have a smaller complexity.

3 Task Representation

To represent the task of finding shortest paths as a reinforcement learning problem, we have to specify $\overline{S}$, $\overline{s}_{start}$, $\overline{G}$, $\overline{A}(s)$ for $s \in \overline{S}$, $\overline{succ}$, and $\overline{r}$. We set $\overline{S} := S$, $\overline{s}_{start} := s_{start}$, $\overline{G} := G$, $\overline{A}(s) := A(s)$ for $s \in \overline{S}$, and use the state transition function $\overline{succ} := succ$, except that we let the lifetime of the agent in formula 1 end when it reaches a goal state. Formally,

$$
\begin{array}{rcll}
\overline{S} & := & S & \\
\overline{s}_{start} & := & s_{start} & \\
\overline{G} & := & G & \\
\overline{A}(s) & := & \begin{cases} \{id\} & \text{if } s \in \overline{G} \\ A(s) & \text{otherwise} \end{cases} & \text{for all } s \in \overline{S} \\
\overline{succ}(s,a) & := & \begin{cases} s & \text{if } a = id \\ succ(s,a) & \text{otherwise} \end{cases} & \text{for all } s \in \overline{S} \text{ and } a \in \overline{A}(s) \\
\overline{r}(s,id) & := & 0 & \text{for all } s \in \overline{G}
\end{array}
$$

where id is an identity action (i.e. it leaves the state unchanged).

This leaves only the immediate rewards $\overline{r}(s,a)$ for $s \in S \setminus G := \{s \in S : s \notin G\}$ and $a \in A(s)$ unspecified. Every reward function $\overline{r}$ can be chosen, as long as it has the following property:

$$
gd(s) < gd(s') \Leftrightarrow U^{opt}(s) > U^{opt}(s') \qquad \text{for all } s, s' \in S \tag{3}
$$

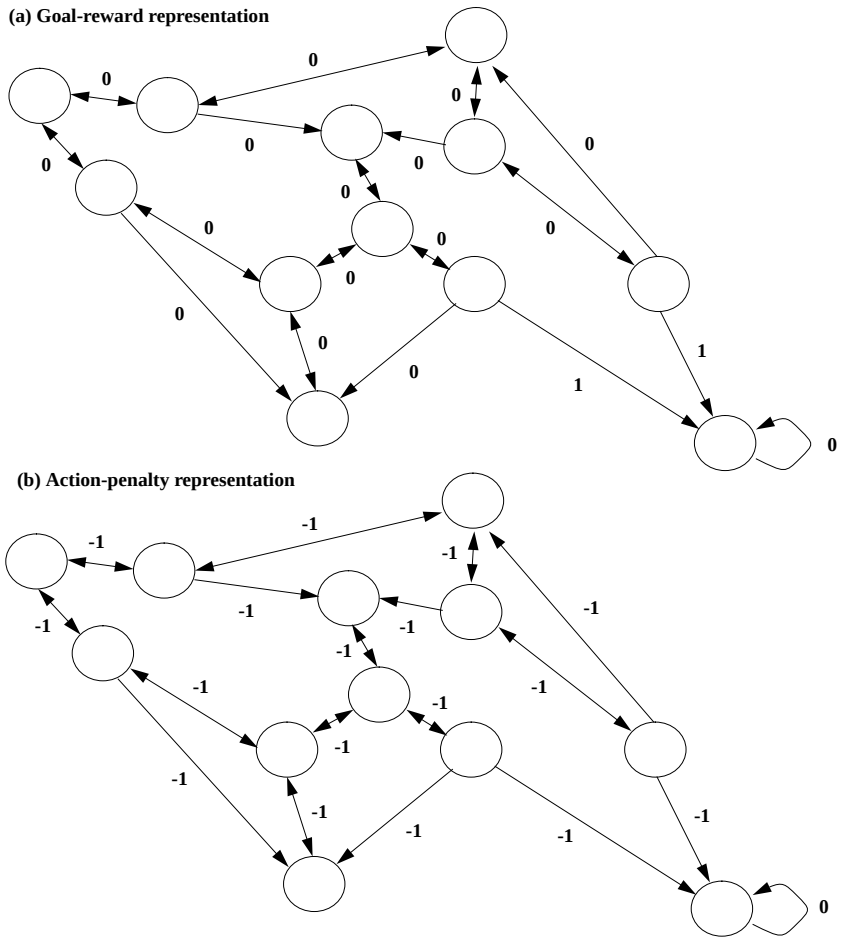

Figure 6: Two representations of the map

i.e. a state with a smaller goal distance has a larger optimal total reward and vice versa. This guarantees that shorter paths to a goal state are preferred over longer paths which in turn are preferred over paths that do not lead to a goal state (i.e. let the agent cycle in non-goal states). If condition 3 did not hold, then reinforcement learning algorithms would prefer longer paths to a goal state over shorter ones, since they maximize the total reward for every state.

We consider two possible reward functions with this property, both shown in Figure 6. (We use the undirected edge $s \leftrightarrow s'$ as a shortcut for the two directed edges $s \rightarrow s'$ and $s \leftarrow s'$.)

3.1 Goal-Reward Representation

In the **goal-reward representation**, the agent is rewarded for entering a goal state, but not rewarded or penalized otherwise. This representation has been used by [35], [33], [31], and [24], among others.

$$\overline{r}(s,a) = \begin{cases} 1 & \text{if } succ(s,a) \in G \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } s \in S \setminus G \text{ and } a \in A(s)$$

The optimal total discounted reward of $s \in S \setminus G$ is $\gamma^{gd(s)-1}$. If no discounting is used, then the optimal total reward is 1 for every $s \in S \setminus G$, independent of its goal distance, since the state space is strongly connected. Thus, discounting is necessary so that shorter goal distances equate with larger optimal total rewards, i.e. in order to satisfy property 3.

3.2 Action-Penalty Representation

In the **action-penalty representation**, the agent is penalized for every action that it executes. This representation has a more dense reward structure than the goal-reward representation (i.e. the agent receives non-zero rewards more often) if goals are relatively sparse. It has been used by [3], [2], and [13], among others.

$$\overline{r}(s,a) = \Leftrightarrow 1 \qquad \text{for } s \in S \setminus G \text{ and } a \in A(s)$$

The optimal total discounted reward of $s \in S$ is $(1 \Leftrightarrow \gamma^{gd(s)})/(\gamma \Leftrightarrow 1)$. Its optimal total undiscounted reward is $\Leftrightarrow gd(s)$. Note that discounting can be used with the action-penalty representation, but is not necessary to satisfy restriction 3. Therefore, the action-penalty representation provides additional freedom when choosing the parameters of the reinforcement learning algorithms. The Q-values and U-values are integers if no discounting is used, otherwise they are reals. Integers have the advantage over reals that they need less memory space and can be stored without loss in precision.

3.3 The Problem of Delayed Rewards

In the goal-reward representation, actions that enter a goal state have a positive immediate reward. All other actions have no immediate reward or penalty at all. Thus, the agent receives its first non-zero reward when it enters a goal state for the first time. In the action-penalty representation, all actions (in non-goal states) have an immediate cost of one. Thus, the agent always receives non-zero rewards no matter which actions it executes.

The problem of delayed rewards is present no matter which of the two representations is used, since it is not necessarily optimal to always execute the action with the largest immediate reward. In fact, in almost all states all actions have the same immediate

reward (provided that goals are relatively sparse), namely 0 if goal-reward representation is used or $\Leftrightarrow$1 if action-penalty representation is used, but usually not all actions are equally preferable.

To summarize, the agent immediately receives non-zero rewards if action-penalty representation is used but no immediate rewards if goal-reward representation is used. However, switching from goal-reward representation to action-penalty representation does *not* transform the reinforcement learning problem from one with delayed rewards to one with immediate rewards and, thus, does not make the credit-assignment problem trivial to solve.

4 Reaching a Goal State with Q-Learning

We can now determine the complexity of reinforcement learning algorithms for the path planning tasks. We first analyze the complexity of reaching a goal state for the first time. The agent does *not* need to find shortest paths. It only has to reach a goal state from the start state. We use the total number of steps that the agent needs to solve the task in the worst-case as a measure for the complexity of the on-line algorithms. The worst-case complexity of reaching a goal state provides a lower bound on the complexity of finding *all shortest* paths, since this cannot be done without knowing where the goal states are. By "worst-case complexity" we mean an upper bound on the number of steps for an uninformed algorithm that holds for all possible topologies of the state space, start and goal states, and tie breaking rules among indistinguishable actions (i.e. actions that have the same Q-values). Clearly, in order to have a worst-case complexity smaller than infinity, an initially uninformed search algorithm must learn something about the effects of action executions.

4.1 Zero-Initialized Q-Learning with Goal-Reward Representation

Assume that a Q-learning algorithm operates on the goal-reward representation and is zero-initialized, i.e. has no initial knowledge of the topology of the state space. Such an algorithm is uninformed.

Definition 1 *A Q-learning algorithm is* **initialized** *with $q \in \mathcal{R}$ (or, synonymously, q-initialized), iff initially*

$$Q(s,a) = \begin{cases} 0 & \text{if } s \in G \\ q & \text{otherwise} \end{cases} \qquad \text{for all } s \in S \text{ and } a \in A(s)$$

An example for a possible behavior of the agent in the state space with the reward structure from Figure 6(a) is shown in Figure 7. It depicts the beginning and end of

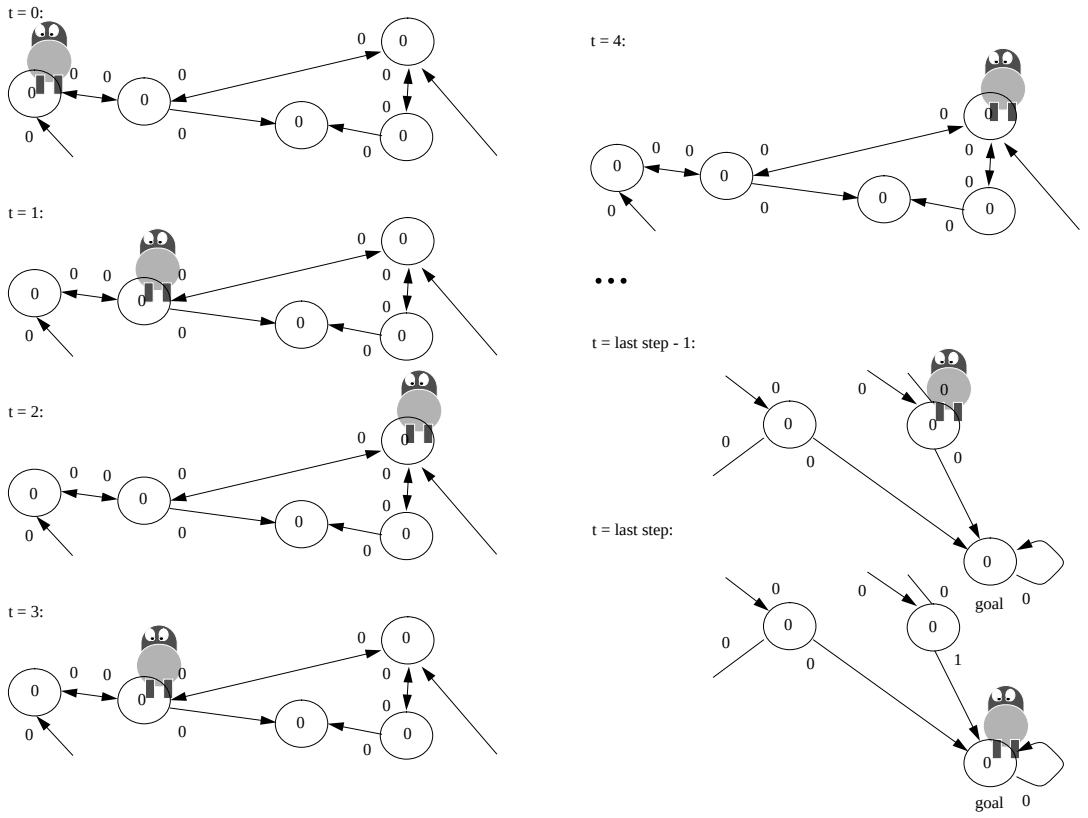

Figure 7: Possible trace of the agent if goal-reward representation is used

a **trace**, i.e. state sequence that the agent traverses. The U-values are shown in the states, and the Q-values label the actions.

During the search for a goal state, all Q-values remain zero. The Q-value of the action that leads the agent to a goal state is the first Q-value that changes. For all other actions, no information about the topology of the state space is remembered by the agent, i.e. it does not remember the effects of actions. In fact, it does not even remember which actions it has already explored. As a consequence, the action selection step has no information on which to base the decision which action to execute next if it utilizes only the Q-values (i.e. if it performs undirected exploration). We assume that the agent selects actions randomly, i.e. according to a uniform distribution.[10] Therefore, the agent performs a random walk. The same behavior can be achieved without maintaining Q-values or any other information, simply by repeatedly selecting one of the available actions at random and executing it.

A random walk in a strongly connected, finite state space reaches a goal state eventually with probability one. The number of steps needed can exceed every given bound (with

[10] If it had a systematic bias, for example always chose the smallest action according to some given ordering, it would not necessarily reach a goal state and could therefore cycle in the state space forever.

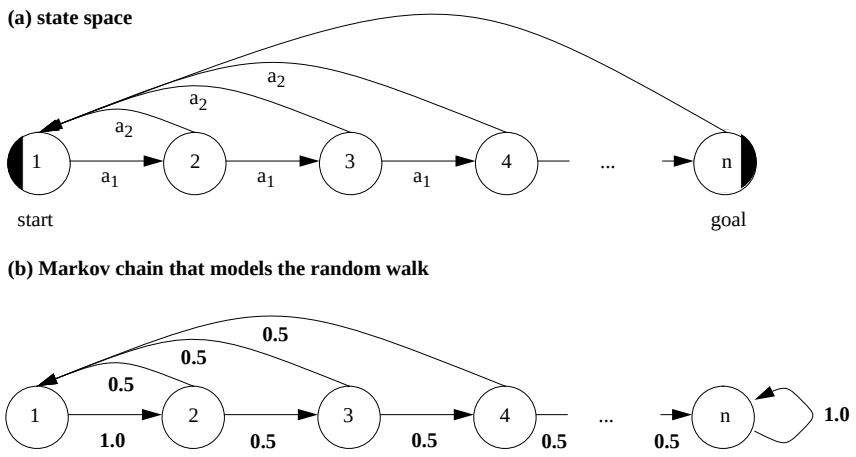

Figure 8: A domain ("reset state space") for which a random walk needs $3 \times 2^{n-2} \Leftrightarrow 2$ steps on average to reach the goal state (for $n \geq 2$)

a probability close to zero) so we use the largest *average* number of steps over all possible topologies of the state space, start and goal states, rather than the worst-case complexity. The number of steps needed on average to reach a goal state from state s equals the absorption time of s in the Markov chain that corresponds to the random walk. (See Figure 8(b) for an example whose transitions are labeled with the transition probabilities.) For every $s \in S$ we introduce a variable $x_s \in \mathcal{R}$ that represents the average number of steps needed until a goal state is reached if the agent starts in s. These values can be calculated by solving the following set of linear equations

$$x_s = \begin{cases} 0 & \text{if } s \in G \\ 1 + \dfrac{1}{|A(s)|} \displaystyle\sum_{a \in A(s)} x_{succ(s,a)} & \text{otherwise} \end{cases} \qquad \text{for all } s \in S \tag{4}$$

$x_{s_{start}}$ can scale exponentially with n, the number of states. Consider for example the reset state space shown in Figure 8. A "reset" state space is one in which all states (except for the start state) have an action that leads back to the start state. It corresponds for example to the task of stacking n blocks if the agent can either stack another block or scramble the stack in every state.

Theorem 1 (Whitehead [37]) *The expected number of steps that a zero-initialized Q-learning algorithm with goal-reward representation needs in order to reach a goal state and terminate can be exponential in n.*

[35] made this observation for state spaces that have the following property: In every state (except for the border states), the probability of choosing an action that leads

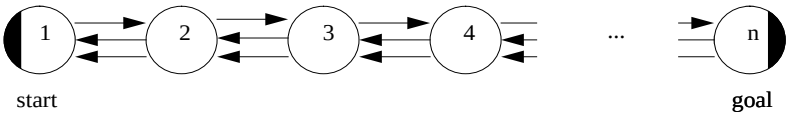

Figure 9: A domain for which a random walk needs $2^{n+1} \Leftrightarrow 3n \Leftrightarrow 1$ steps on average to reach the goal state (for $n \geq 1$)

away from the (only) goal state is larger than the probability of choosing an action that leads closer to the goal state. Consider for example the version of a one-dimensional gridworld shown in Figure 9, that is similar to one in [33]. In every state (but the border states) the agent can execute three actions: one leads to the right, the other two are identical and lead to the left. If an action is chosen randomly, chances are 2/3 that the agent goes to the left (i.e. goes away from the goal state) and only 1/3 that it goes to the right (i.e. approaches the goal state).

This observation motivated [35] to explore cooperative reinforcement learning algorithms in order to decrease the complexity. [32] showed that even non-cooperative reinforcement learning algorithms have polynomial worst-case complexity if reinforcement learning is augmented with a directed exploration mechanism that he calls "counter-based Q-learning." We will show that one does not need to augment Q-learning. It is tractable if one uses either the action-penalty representation or different initial Q-values. This confirms experimental observations of [11], [36], [19], and [24], who mention an improvement in performance during their experiments when using the action-penalty representation or initializing the Q-values high instead of using the goal-reward representation and initializing the Q-values with zero.

4.2 Using a Different Task Representation

Assume now that we are still using a zero-initialized Q-learning algorithm, but let it operate on the action-penalty representation. Although the algorithm is still uninformed, the Q-values change immediately, starting with the first action execution, since the reward structure is dense. In this way, the agent remembers the effects of previous action executions. An example for a possible behavior of the agent in the state space with the reward structure from Figure 6(b) is shown in Figure 10. The U-values are shown in the states, and the Q-values label the actions.

4.2.1 Complexity Analysis

While we state the following definitions, lemmas, and theorems for the case in which no discounting is used, they can easily be adapted to the discounted case as outlined

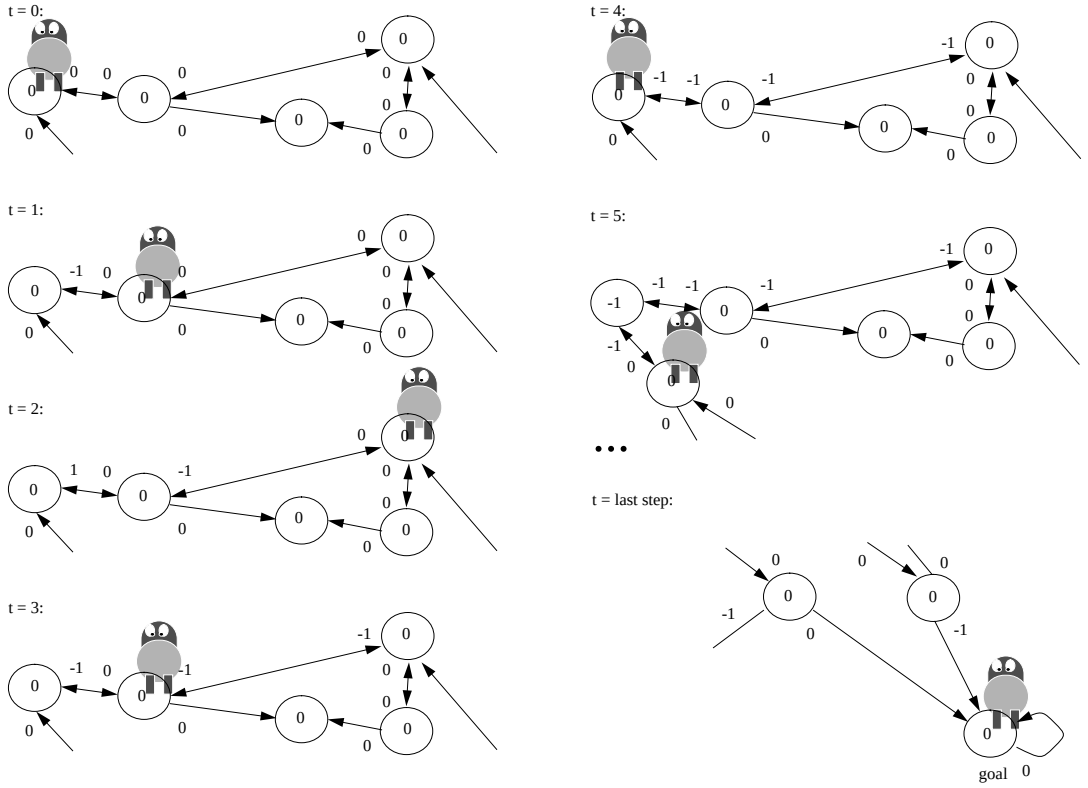

Figure 10: Possible trace of the agent if action-penalty representation is used ($\gamma = 1$)

later in this chapter.

Definition 2 *Q-values are* **consistent** *iff*

$$\left.\begin{array}{r} 0 \\ \Leftrightarrow 1 + U(succ(s,a)) \end{array}\right\} \leq Q(s,a) \leq 0 \quad \left\{\begin{array}{l} for\ all\ s \in G\ and\ a \in A(s) \\ for\ all\ s \in S \setminus G\ and\ a \in A(s) \end{array}\right.$$

Zero-initialized Q-values are consistent.

Definition 3 *Q-values are* **admissible** *iff*

$$\left.\begin{array}{r} 0 \\ \Leftrightarrow 1 \Leftrightarrow gd(succ(s,a)) \end{array}\right\} \leq Q(s,a) \leq 0 \quad \left\{\begin{array}{l} for\ all\ s \in G\ and\ a \in A(s) \\ for\ all\ s \in S \setminus G\ and\ a \in A(s) \end{array}\right.$$

Consistent Q-values are admissible.

To understand our choice of terminology, note the following relationship:

$$Q^{opt}(s,a) = \left\{\begin{array}{ll} 0 & \text{for all } s \in G \text{ and } a \in A(s) \\ \Leftrightarrow 1 + U^{opt}(succ(s,a)) = \Leftrightarrow 1 \Leftrightarrow gd(succ(s,a)) & \text{for all } s \in S \setminus G \text{ and } a \in A(s) \end{array}\right.$$

since $U^{opt}(s) = \Leftrightarrow gd(s)$ for all $s \in S$. Thus, Q-values are admissible iff $Q^{opt}(s,a) \leq Q(s,a) \leq 0$ for all $s \in S$ and $a \in A(s)$. This parallels the notion of admissibility that is used for heuristic estimates in connection with A*-search (see for example [20] or [22]). Furthermore, if a heuristic h for the goal distance is known that is admissible for A*-search, then the following Q-values are admissible as well

$$Q(s,a) = \begin{cases} 0 & \text{for all } s \in G \text{ and } a \in A(s) \\ \Leftrightarrow 1 \Leftrightarrow h(succ(s,a)) & \text{for all } s \in S \setminus G \text{ and } a \in A(s) \end{cases}$$

Admissible heuristics are known for many search problems in Artificial Intelligence, for example for path planning problems in a gridworld (or more general: on a topological map) or solving the 8-puzzle.

Definition 4 *A Q-learning algorithm is* **admissible** *iff it uses the action-penalty representation, its action selection step is "$a := argmax_{a' \in A(s)} Q(s,a')$," and either*

- *its initial Q-values are consistent, and its value update step is "Set $Q(s,a) := \Leftrightarrow 1 + U(succ(s,a))$,"*[11] *or*
- *its initial Q-values are admissible, and its value update step is "Set $Q(s,a) := \min(Q(s,a), \Leftrightarrow 1 + U(succ(s,a)))$."*

If a Q-learning algorithm is admissible, then consistent (admissible) Q-values remain consistent (admissible) after every step of the agent and are monotonically decreasing.

The action selection step of an admissible Q-learning algorithm always executes the action with the largest Q-value. This strategy avoids the exploration exploitation conflict, since it always exploits (i.e. executes the action that currently seems to be best) but at the same time explores sufficiently often (i.e. executes actions that it has never executed before). It performs undirected exploration, since it uses only the Q-values for action selection and no other information.

Call $PG := \{s \in S : U(s) = 0\} \supseteq G$ the set of potential goal states. If the Q-learning algorithm is zero-initialized, then PG is always the set of states in which the agent has not yet explored all of the actions. We call an action **explored** iff the agent has executed it at least once. If Q-values are consistent, then

$$\Leftrightarrow 1 \Leftrightarrow \min_{s' \in PG} d(succ(s,a), s') \leq Q(s,a) \leq 0 \qquad \text{for all } s \in S \setminus G \text{ and } a \in A(s)$$

The action selection step can then be interpreted as using $Q(s,a)$ to approximate $\Leftrightarrow 1 \Leftrightarrow \min_{s' \in PG} d(succ(s,a), s'))$ and tending (sometimes unsuccessfully) to direct the agent from the current state to the closest potential goal state with as few steps as possible and make it then take an unexplored action.

[11] i.e. "Set $Q(s,a) := \overline{r}(s,a) + \gamma U(succ(s,a))$," where $\overline{r}(s,a) = -1$ and $\gamma = 1$.

It is easy to see that an admissible Q-learning algorithm eventually reaches a goal state. The argument parallels a similar argument for RTA*-type algorithms [17, 25] and is by contradiction: If the agent did not reach a goal state eventually, it would run around in a cycle that would not include a goal state. For every cycle that the agent completed, the largest Q-value of the actions executed in the cycle would decrease by at least one. Eventually, the Q-values of all actions executed in the cycle would drop below every bound. In particular, they would drop below the Q-value of an action that, when executed, would make the agent leave the cycle. Such an action exists, since the state space is strongly connected and thus a goal state can be reached from within the cycle. Then, the agent would leave the cycle, which is a contradiction.

Now that we know that the algorithm terminates (which also means that the algorithm is correct, i.e. reaches a goal state), we are interested in its complexity. Lemma 1 contains the central invariant for all proofs. It states that the number of steps executed so far is always bounded by an expression that depends only on the initial and current Q-values and, more over, "that the sum of all Q-values decreases (on average) by one for every step taken" (this paraphrase is grossly simplified). A time superscript of t in Lemmas 1 and 2 refers to the values of the variables immediately before the action execution step, i.e. line 4, of step t.

Lemma 1 *For all steps $t \in \mathcal{N}_0$ (until termination) of an undiscounted, admissible Q-learning algorithm, it holds that*

$$U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) \Leftrightarrow t \geq \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) + U^0(s^0) \Leftrightarrow loop^t$$

and

$$loop^t \leq \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) \Leftrightarrow \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a),$$

where $loop^t := |\{t' \in \{0, \ldots, t \Leftrightarrow 1\} : s^{t'} = s^{t'+1}\}|$ (the number of identity actions, i.e. actions that do not change the state, executed before t).

Lemma 2 *An undiscounted, admissible Q-learning algorithm reaches a goal state and terminates after at most*

$$2 \sum_{s \in S \setminus G} \sum_{a \in A(s)} (Q^0(s,a) + gd(succ(s,a)) + 1) \Leftrightarrow U^0(s^0)$$

steps.

Theorem 2 *An admissible Q-learning algorithm reaches a goal state and terminates after at most $O(ed)$ steps.*

Lemma 2 utilizes the invariant and the fact that each of the e different Q-values is bounded by an expression that depends only on the goal distances to derive a bound

on t. Since "the sum of all Q-values decreases (on average) by one for every step taken" according to the invariant, but is bounded from below, the algorithm must terminate. Since $gd(s) \leq d$ for all $s \in S$, the result from Theorem 2 follows directly. $O(ed) \leq O(en)$, since $d \leq n \Leftrightarrow 1$ in every strongly connected state space. (Remember that n denotes the number of states, e the total number of actions, and d the depth of the state space.) Thus, an admissible Q-learning algorithm reaches a goal state and terminates after at most $O(en)$ steps. This worst-case performance provides, of course, an upper bound on the average-case performance of the algorithm.

Theorem 2 provides an upper bound on the complexity of every admissible Q-learning algorithm, including a zero-initialized algorithm. To demonstrate that $O(en)$ is a tight bound for a zero-initialized Q-learning algorithm, we show that it is also a lower bound. Lower bounds can be proved by example, i.e. by showing a domain for which the Q-learning algorithm can need this many steps to reach a goal state. Such a domain is depicted in Figure 11. Thus, the worst-case complexity of reaching a goal state with admissible, zero-initialized Q-learning is tight at $O(en)$.

4.2.2 Comparison of Q-Learning to other On-Line Search Algorithms

We define an (initially) **uninformed on-line search algorithm** to be an algorithm that does not know the effect of an action (i.e. does not know which successor state the action leads to) before it has executed it at least once. One example of such an algorithm is zero-initialized Q-learning.

A highly **reactive** algorithm, such as Q-learning, often executes actions that are sub-optimal (when judged according to the knowledge that the agent could have acquired if it had memorized all of its experiences). For example, the agent can move around for a long time in parts of the state space that it has already explored, thus neither exploring unknown parts of the state space nor having a chance to find a goal. This can be avoided when **planning** further into the future: the agent learns a model of the state space ("world model", which is in our case a map), and uses it to predict the effects of action executions. This enables the agent to make a more informed decision about which action to execute next.

The DYNA architecture [30] [31], for example, implements look-ahead planning in the framework of Q-learning. Actions are executed in the real world mainly to refine (and, in non-stationary environments, to update) the model. The model is used to simulate the execution of actions and thereby to create experiences that are indistinguishable from the execution of real actions. This way, the real world and the model can interchangeably be used to provide input for Q-learning. Using the model, the agent can optimize its behavior according to its current knowledge without having to execute actions in the real world. Also, the agent can simulate the execution of arbitrary (not just local) actions at any time. Various researchers, for example [24] and [19], have devised strategies which actions to simulate in order to speed-up planning.

There is a trade-off: When learning a map and using it for planning, the agent has

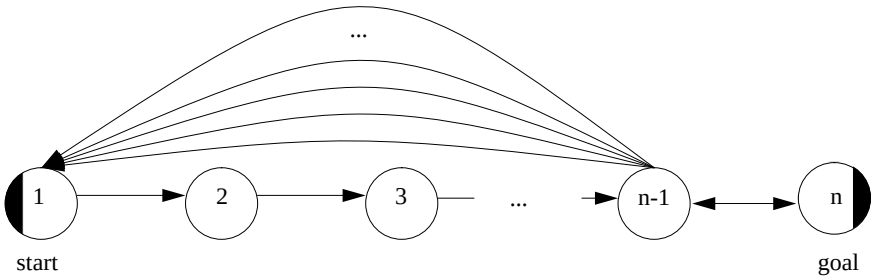

Figure 11: A domain for which an admissible, zero-initialized Q-learning algorithm (and every other uninformed on-line search algorithm) can need at least $(e \Leftrightarrow n + 1)(n \Leftrightarrow 1)$ steps to reach the goal state (for $n \geq 2$ and $e \geq n$)

to keep more information around and perform more computations between action executions, which increases its deliberation time between action executions. However, chances are that the agent needs less steps to reach a goal state. In the following, we compare the worst-case complexity of the Q-learning algorithm, that does not learn a map, to the worst-case complexity of any other uninformed search algorithm, e.g. one that learns a map and uses it for planning.

Figure 11 shows that *every* uninformed on-line search algorithm has a worst-case complexity of at least $O(en)$. If ties are broken in favor of actions that lead to states with smaller numbers, then every uninformed on-line search algorithm can traverse a supersequence of the following state sequence: $e \Leftrightarrow n + 1$ times the sequence $123 \ldots n \Leftrightarrow 1$, and finally n. Basically, all of the $O(e)$ actions in state $n \Leftrightarrow 1$ are executed once. All of these lead the agent back to state 1 and therefore force it to execute $O(n)$ explored actions before it can explore a new action. Thus, the worst-case complexity is at least $O(en)$.

We had already seen that one could decrease the complexity of Q-learning dramatically (i.e. in some cases exponentially) by choosing the action-penalty representation over the goal-reward representation. Now, we have shown that every uninformed on-line search algorithm has at least the same worst-case complexity as a zero-initialized Q-learning algorithm with action-penalty representation. This does not mean, however, that such a Q-learning algorithm is the best possible algorithm for reaching a goal state. In the following, we show that there exists an uninformed on-line search algorithm that strictly dominates an admissible, zero-initialized Q-learning algorithm. We say that an algorithm X **strictly dominates** an algorithm Y if X always performs no worse (i.e. needs no more steps) than Y and performs strictly better in at least one case.

Consider an algorithm that maintains a map of the part of the state space that it has explored so far. It executes unexplored actions in the same order as zero-initialized Q-learning, but always chooses the shortest known path to the next unexplored action: The agent uses its model of the world, i.e. the (partial) map and its knowledge of the

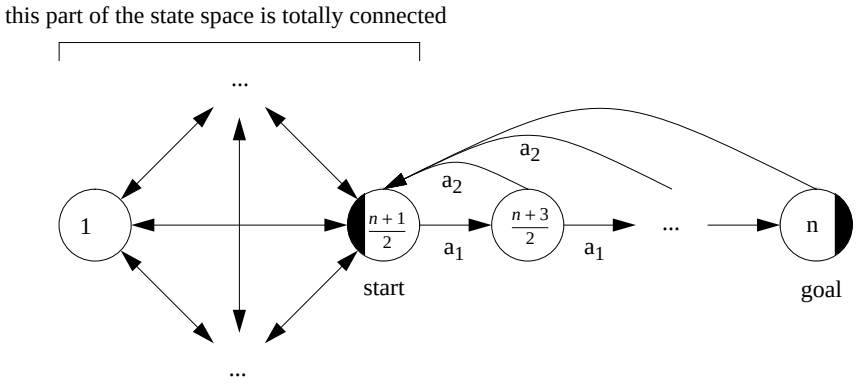

Figure 12: A domain for which an admissible, zero-initialized Q-learning algorithm can need at least $1/16n^3 \Leftrightarrow 3/16n^2 \Leftrightarrow 1/16n + 3/16$ steps to reach the goal state, but Q_{map}-learning needs only at most $3/8n^2 + 3/2n \Leftrightarrow 23/8$ steps (for odd $n \geq 3$)

Q-values, to simulate its behavior under the Q-learning algorithm until it would execute an unexplored action. Then, it uses the map to find the shortest known action sequence that leads from its current state in the world to the state in which it can execute this action, executes the action sequence and the unexplored action, and repeats the cycle. We call this algorithm the **Q_{map}-learning algorithm**. Per construction, it cannot perform worse than zero-initialized Q-learning (no matter what the tie-breaking rule is) if ties are broken in the same way . Thus, the worst-case complexity of Q_{map}-learning over all tie-breaking rules cannot be worse than the one of Q-learning. Consider, for example, the state sequence that Q-learning traverses in a reset state space (shown in Figure 8) of size $n = 6$ if ties are broken in favor of actions that lead to states with smaller numbers: 121231234123451234123456. First, Q-learning finds out about the effect of action a_1 in state 1 and then about a_2 in 2, a_1 in 2, a_2 in 3, a_1 in 3, a_2 in 4, a_1 in 4, a_2 in 5, and a_1 in 5, in this order. The Q_{map}-learning algorithm explores the actions in the same order. However, after it has executed action a_2 in state 5 for the first time, it knows how to reach state 5 again faster than Q-learning: it goes from state 1 through states 2, 3, and 4, to state 5, whereas Q-learning goes through states 2, 1, 2, 3, and 4. Thus, the Q_{map}-learning algorithm traverses the following state sequence: 121231234123451234123456, and is two steps faster than Q-learning. Figure 12 gives an example of a domain for which the big-O worst-case complexities of the two algorithms are different: There is a tie-breaking rule that causes Q-learning to need $O(n^3)$ steps to reach the goal state, whereas Q_{map}-learning needs at most $O(n^2)$ steps no matter how ties are broken. In short: Q-learning can require $O(n^3)$ steps to reach the goal state, whereas Q_{map}-learning reaches the goal state with at most $O(n^2)$ steps.

The Q_{map}-learning algorithm is mainly of theoretical interest, because it demonstrates that there exist algorithms that dominate Q-learning, and the domination proof is easy. However, algorithms whose behavior resembles the one of the Q_{map}-learning algorithm

are not only of theoretical interest:

One idea behind the DYNA architecture is that executing actions in the real world is slow (and expensive), whereas simulating the execution of actions in a model of the world is fast (and inexpensive). Therefore, planning should exclusively be done in the model if possible. Once an action is explored in the real world, it can be integrated in the model. The effect of an action does not change over time, since the state space is stationary. Consequently, actions should only be executed in the real world

- to learn the effect of the executed (unexplored) action,
- to get the agent into a state in which it can find out about the effect of an unexplored action, or
- to get the agent to a goal.

If executing actions in the real world and simulating action executions in the model take approximately the same time, then there is a trade-off. Simulating actions allows the agent to utilize its current knowledge better, whereas executing actions in the real-world increases its knowledge and allow it to stumble across a goal. This trade-off has for example been investigated by [15] and [16]. Using the model provides the agent with the advantage that all actions can be simulated at any time, whereas the world constrains it to execute only actions in its current state. Reinforcement learning researchers usually assume that the agent can simulate x action executions for every action execution in the real world. Then, the problem arises which actions to simulate. Reinforcement learning researchers have proposed real-time schemes for planning in the world model of a DYNA architecture that are used to approximate the following behavior of the agent: "If the current world state is a goal state, stop. Otherwise, go to the closest state with an unexplored action, execute it, and repeat." This way, one prevents the agent from unnecessarily executing actions that it has already explored, which is also the objective of the Q_{map}-learning algorithm. Different researchers update the Q-values in the model in different orders. For example, [23] and [9] study reinforcement learning algorithms that have a look-ahead larger than one or perform best-first search. In contrast, [19] updates those Q-values in the model first that are expected to change most. Note that we have not included the planning time in the complexity measure. If planning time is not negligible compared to execution time, then the total run-time (i.e. the sum of planning and execution time) can increase even if the number of steps needed (i.e. the execution time) decreases.

The relationships between an admissible, zero-initialized Q-learning algorithm, the Q_{map}-learning algorithm, and uninformed on-line search algorithms in general are summarized in Figure 13 (for domains that have no duplicate actions, see Chapter 4.4.1). They show that it can be misleading to focus only on the worst-case complexity of an on-line search algorithm over all domains. We have demonstrated that there exists an uninformed on-line search algorithm, namely Q-learning, that reaches a goal state in $O(n^3)$ steps no matter what the domain is. There are domains in which no uninformed on-line search algorithm can do better. However, there exists an uninformed

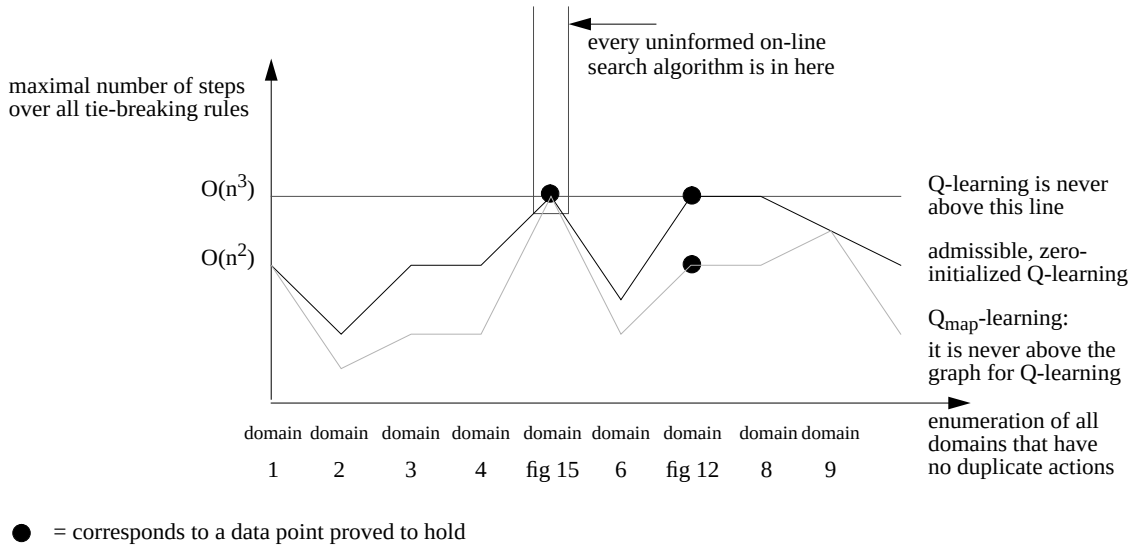

Figure 13: A diagram showing the relationships between an admissible, zero-initialized Q-learning algorithm, the Q_{map}-learning algorithm, and uninformed on-line search algorithms in general (data points for domains other than the ones shown in Figures 12 and 15 are fictitious)

on-line search algorithm, namely Q_{map}-learning, that always performs no worse than Q-learning, but reduces the number of action executions by more than a constant factor in at least one domain. To summarize, although every uninformed on-line search algorithm has at least the same big-O worst-case complexity as an admissible, zero-initialized Q-learning algorithm, learning a map of the state space (and subsequently using it for planning) can decrease the big-O worst-case complexity for *some* (but not all) domains.

4.2.3 Comparison of Q-Learning to Off-Line Search Algorithms

The ratio of the worst-case effort spent by a given on-line algorithm to solve a given problem and the worst-case effort spent by the best informed off-line algorithm to solve the same problem is called the **competitive ratio** of the on-line algorithm. An on-line algorithm is called **competitive** if its competitive ratio is bounded from above by a constant (i.e. the upper bound is independent of the problem size) [29]. Figure 14 shows, as expected, that *no* uninformed on-line search algorithm is competitive: they need $O(en)$ steps in the worst case, but an off-line algorithm that knows the topology of the state space can reach the goal state in only one step.[12] On-line search algorithms

[12] One could argue that it would be more appropriate to use the maximum of the ratio of the *average-case* effort spent by the on-line algorithm and the *average-case* effort spent by the best totally informed off-line algorithm. For the path planning tasks, the average-case complexity of the off-line algorithm

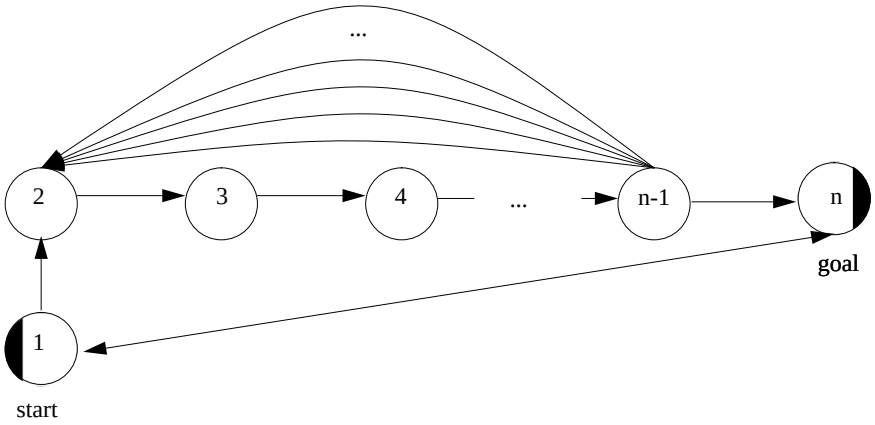

Figure 14: A domain for which an admissible, zero-initialized Q-learning algorithm (and every other uninformed on-line search algorithm) can need at least $(e \Leftrightarrow n)(n \Leftrightarrow 2) + 1$ steps to reach the goal state, but an off-line search algorithm that knows the topology of the state space needs only one step to reach the goal state (for $n \geq 3$ and $e \geq n + 1$)

with a look-ahead of one step (such as Q-learning), that reduce deliberation between action executions to a minimum, and off-line search algorithms represent extreme points on the deliberation action scale, with on-line search algorithms that have a limited look-ahead larger than one in between. The competitive ratio of $O(en)$ demonstrates that one increases the number of steps executed by a factor of $O(en)$ at most when moving from the reactive extreme of the deliberation action scale to the deliberative extreme.

4.2.4 Discounting

All properties of an undiscounted, admissible Q-learning algorithm can immediately be transferred to the case in which discounting is used, since there is a strictly monotonically increasing bijection between the Q-values of the former algorithm after step t and the Q-values of the latter algorithm after the same step, namely: a Q-value of $Q^t(s,a) = q \in \mathcal{N}_0^-$ in the undiscounted case corresponds to a Q-value of $Q^t(s,a) = (1 \Leftrightarrow \gamma^{-q})/(\gamma \Leftrightarrow 1)$ in the discounted case. Since both algorithms always execute the action with the largest Q-value, they always choose the same action for execution (if ties are broken in the same way). Thus, they behave identically.

equals its worst-case complexity, since action outcomes are deterministic. The complexity of the on-line algorithm, to the contrary, depends on how ties are broken, which is a random process. Therefore, its average-case complexity can be smaller than its worst-case complexity. However, Figure 14 also shows that no uninformed on-line search algorithm is competitive, even under this relaxed definition of competitiveness. We continue to use our original definition, since we are for now more interested in the worst-case complexity of the algorithms than in their average-case performance.

4.3 Using Different Initial Q-Values

We now analyze Q-learning algorithms that operate on the goal-reward representation, but are one-initialized. A similar initialization has been used before, for instance in experiments conducted by [11]. It assumes no prior knowledge of the topology of the state space if one makes the reasonable assumption that the value update step knows after the execution of an action whether the new state of the agent is a goal state or not. Then, the Q-values of actions in non-goal states can be initialized differently from the Q-values of actions in goal states. We assume (again) that the algorithm uses an action selection step that executes the action with the largest Q-value.

If the value update step knows whether the new state of the agent is a goal state, one can set the initial Q-values of actions in non-goal states to -1 and the ones of actions in goal states to 0. Assigning -1 to non-goal states reflects the fact that the agent knows that the state is a non-goal state once it is in the state and therefore can conclude that it takes at least one action execution to reach a goal state. Such an undiscounted, (minus one)-initialized Q-learning algorithm with action-penalty representation behaves identically to a discounted one-initialized Q-learning algorithm with goal-reward representation, since there is a strictly monotonically increasing bijection between the Q-values of the former algorithm after step t and the Q-values of the latter algorithm after the same step, namely: A Q-value of $Q^t(s,a) = q \in \mathcal{N}^-$ $(Q^t(s,a) = 0)$ for a (minus one)-initialized Q-learning algorithm with action-penalty representation corresponds to a Q-value of $Q^t(s,a) = \gamma^{-q-1}$ $(Q^t(s,a) = 0)$ for a one-initialized Q-learning algorithm with goal-reward representation.[13] (As argued in Chaper 3.1, discounting is necessary if goal-reward representation is used.) Since both algorithms always execute the action with the largest Q-value, they always choose the same action for execution (if ties are broken in the same way). Thus, they behave identically. Since (minus one)-initialized Q-values are consistent, Theorem 2 applies to the former algorithm. This leads to the following conclusion.

Theorem 3 *A discounted, one-initialized Q-learning algorithm with goal-reward representation reaches a goal state and terminates after at most $O(ed)$ steps if its action selection step is "$a := argmax_{a' \in A(s)} Q(s,a')$" and its value update step is "Set $Q(s,a) := \overline{r}(s,a) + \gamma U(succ(s,a))$."*

The other results and remarks from the previous chapter can be transferred as well.

[13] This argument generalizes to the bi-directional Q-learning algorithm, that is stated later in this report. If the task is only to reach a goal state and stop (i.e. the task that we discuss in this chapter), then a *zero-initialized* Q-learning algorithm with action-penalty representation also behaves identically to a discounted, one-initialized Q-learning algorithm with goal-reward representation. In this case, it does not matter how the Q-values of actions in goal states are initialized.

4.4 Gridworlds and other Common Reinforcement Learning Domains

Gridworlds and other domains studied in the context of reinforcement learning can have special properties such as the following.

Definition 5 *A state space has* **no identity actions** *iff, for all* $s \in S$ *and* $a \in A(s)$, $succ(s,a) \neq s$, *i.e. there are no actions that leave the state unchanged.*

If the agent knows that an action is an identity action, it does not need to execute it, since it is never optimal to execute an identity action. However, according to our assumptions the agent is not aware of the fact whether an action is an identity action. For example, sometimes it is assumed that the location of an agent in a gridworld does not change when the agent tries to move forward, but bumps into an obstacle. Although the absence of identity actions can potentially simplify the path planning tasks, all of the big-O worst-case complexities stated in this report remain unchanged if the state space has no identity actions. Thus, all of the state spaces that we use as examples (except for the one in Figure 22) have no identity actions. However, other properties of reinforcement learning domains can decrease the big-O worst-case complexity of reaching a goal state with Q-learning. In the following, we will identify such properties.

4.4.1 State Spaces with no Duplicate Actions, Constant Individual Upper Action Bounds, Linear Total Upper Action Bounds, or Polynomial Widths

Consider the following four properties of state spaces:

Definition 6 *A state space has* **no duplicate actions** *iff, for all* $s \in S$ *and* $a, a' \in A(s)$, *either* $a = a'$ *or* $succ(s,a) \neq succ(s,a')$, *i.e. no two actions that are applicable in the same state have the same effect.*[14]

Definition 7 *A state space topology has a* **constant individual upper action bound** $b \in \mathcal{R}$ *iff* $|A(s)| \leq b$ *for all* $s \in S$ *and all* $n \in \mathcal{N}$, *i.e. the number of actions that are applicable in a state is bounded from above by a constant.*

Definition 8 *A state space topology has a* **linear total upper action bound** $b \in \mathcal{R}$ *iff* $e \leq bn$ *for all* $n \in \mathcal{N}$, *i.e. the total number of actions increases at most linearly in the number of states.*

[14] If the agent knows that two or more actions achieve the same effect, it needs to consider only one of them. However, according to our assumptions it is not aware of the fact whether two actions have the same effect.

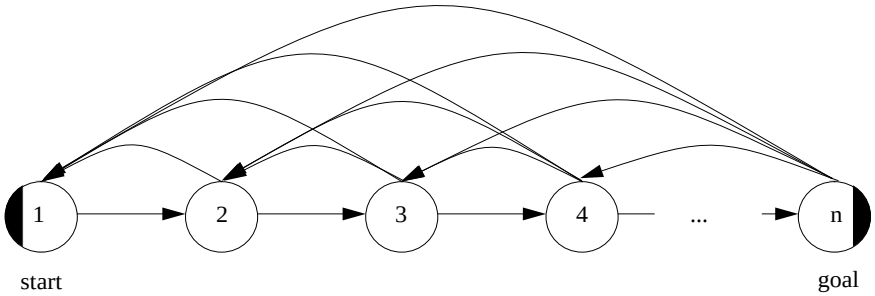

Figure 15: A domain for which an admissible, zero-initialized Q-learning algorithm (and every other uninformed on-line search algorithm) can need at least $1/6n^3 - 1/6n$ steps to reach the goal state (for $n \geq 1$)

Note that every state space topology that has constant individual upper action bound b also has linear total upper action bound b.

Definition 9 *A state space topology has* **polynomial width** *[35] iff there exists a polynomial function p such that $n \leq p(d)$ for all $n \in \mathcal{N}$, i.e. the number of states is a polynomial function of the depth of the state space.*

The following inequalities allow us to express $O(ed)$ in terms of n or d alone by providing an upper bound on e that depends only on n or d: $e \leq n^2$ for state spaces that have no duplicate actions, and $e \leq bn$ for state spaces that have linear total upper action bound b. To be more precise: $O(e) = O(n)$ if the state space has linear total upper action bound b, since then $n \leq e \leq bn$. Similarly, $e \leq p(d)$ (where p is a polynomial function) for state spaces with polynomial width that either have no duplicate actions or a linear total upper action bound.

If a state space has no duplicate actions, then $O(ed) \leq O(n^3)$, and the worst-case complexity of an admissible Q-learning algorithm becomes $O(n^3)$. This bound is tight for a zero-initialized Q-learning algorithm, as shown in Figure 15. This demonstrates that, although Q-learning performs undirected exploration, its worst-case complexity for reaching a goal state is polynomial (and no longer exponential) in n if the action-penalty representation is chosen instead of the goal-reward representation. Again, *every* uninformed on-line search algorithm has at least the same big-O worst-case complexity as zero-initialized Q-learning.

If a state space topology has linear total upper action bound b, then the worst-case complexity becomes $O(ed) \leq O(bn^2) = O(n^2)$. If it has polynomial width and either no duplicate actions or a linear total upper action bound, then $e \leq p(d)$ for a polynomial function p and the complexity becomes $O(ed) \leq O(p'(d))$, where p' is a polynomial function. If the depth d of a state space topology is bounded from above by a constant (i.e. the upper bound is independent of n), then $O(ed) = O(e)$. If it also has no duplicate actions, then the worst-case complexity decreases to $O(ed) = O(e) \leq O(n^2)$.

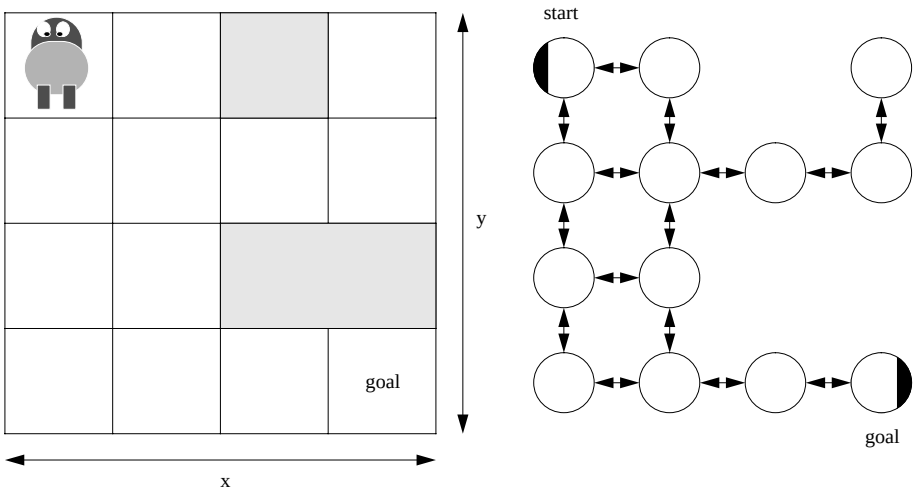

Figure 16: An example of a two-dimensional gridworld

If a state space topology has both a depth with a constant upper bound and a linear total upper action bound, then $O(ed) = O(e) = O(n)$.

Deterministic gridworlds with discrete states, in the following called gridworlds, which have often been used in studying reinforcement learning, see for example [3], [28], [31], [33], [37], or [24], have both a constant individual upper action bound and polynomial width. In the state space shown in Figure 16, for example, the agent can move from any square to each of its four neighboring squares as long as it stays on the grid and the target square does not contain an obstacle (which are shaded in the figure). Consider now such a rectangular gridworld with size $x \times y$, but without obstacles. It has $n = xy$ states, $e = 4xy - 2x - 2y$ actions, and depth $d = x + y - 2$. It has no identity or duplicate actions, a constant individual upper action bound of four and polynomial width, since $n = xy \leq (x + y)^2 = (d + 2)^2$. It holds that $O(ed) = O(x^2y + xy^2)$. If the gridworld is quadratic, i.e. $x = y$, then the depth increases sublinearly in n, since $d = 2\sqrt{n} - 2$, and $O(ed) = O(x^3) = O(n^{3/2})$. Therefore, exploration in unknown gridworlds actually has very low complexity.

4.4.2 State Spaces that are 1-Step Invertible or Eulerian

Gridworlds often have another special property.

Definition 10 *A state space is* **1-step invertible** *[35] iff it has no duplicate actions and, for all $s \in S$ and $a \in A(s)$, there exists an $a' \in A(succ(s,a))$ such that $succ(succ(s,a),a') = s$, i.e. the effect of an action can be reversed immediately.*

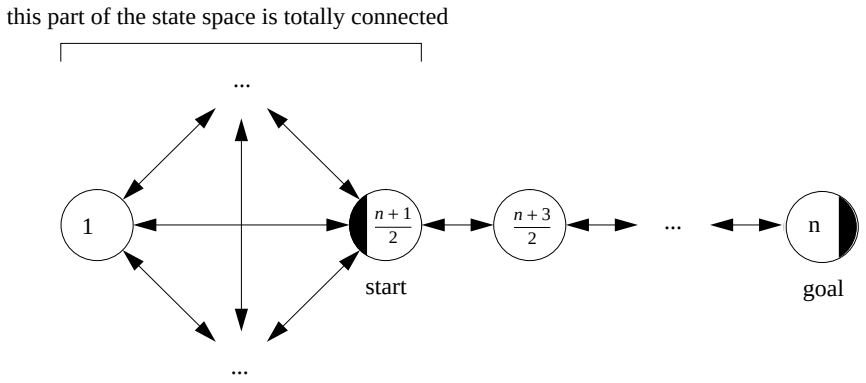

Figure 17: A domain for which a random walk needs $1/8n^3 + 1/8n^2 - 5/8n + 3/8$ steps on average to reach the goal state (for odd $n \geq 1$) and for which an admissible, zero-initialized Q-learning algorithm can need at least $1/16n^3 + 3/8n^2 - 3/16n - 1/4$ steps to reach the goal state (for $n \geq 1$ with $n \bmod 4 = 1$)

Definition 11 *A state space is* **Eulerian** *iff* $|A(s)| = |\{(s', a') : succ(s', a') = s \wedge s' \in S \wedge a' \in A(s')\}|$ *for all* $s \in S$*, i.e. there are as many actions that enter a state as there are actions that leave the state.*

Gridworlds are usually 1-step invertible, and every 1-step invertible state space is Eulerian.

We do *not* assume that the agent knows that the state space is 1-step invertible or Eulerian. Even a zero-initialized Q-learning algorithm with goal-reward representation (i.e. a random walk) is tractable for Eulerian state spaces, as the following theorem states.

Theorem 4 *A zero-initialized Q-learning algorithm with goal-reward representation reaches a goal state and terminates after at most* $O(ed)$ *steps on average if the state space is Eulerian.*

This theorem is an immediate corollary to [1]. $O(ed) \leq O(en)$, since $d \leq n-1$ in every strongly connected state space. Thus, a zero-initialized Q-learning algorithm with goal-reward representation reaches a goal state and terminates after at most $O(en)$ steps on average if the state space is Eulerian. If the state space also has no duplicate actions, then the largest average-case complexity becomes $O(n^3)$. Figure 17 shows that this bound is tight.

As an example of a gridworld, consider the one-dimensional gridworld in Figure 18. The average number of steps that a random walk needs to reach the goal state of this domain is a standard result in Operations Research for a symmetric random walk in

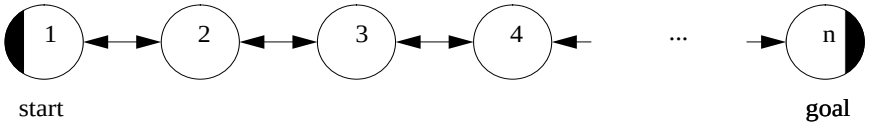

Figure 18: A domain ("one-dimensional gridworld") for which every search algorithm needs at least $n-1$ steps to reach the goal state, and a random walk needs n^2-2n+1 steps on average to reach the goal state (for $n \geq 1$)

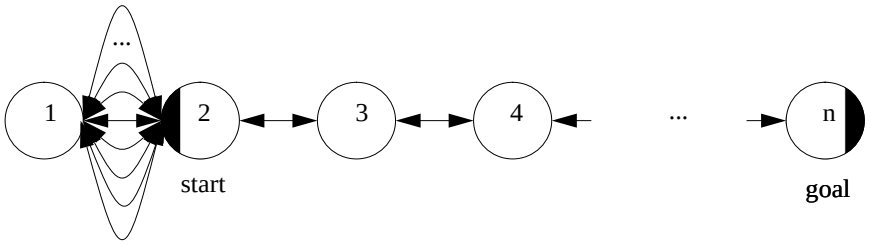

Figure 19: A domain for which every uninformed on-line search algorithm can need at least $e+n-4$ steps to reach the goal state (for $n > 2$ and $e \geq 2n-2$) and for which an admissible, zero-initialized Q-learning algorithm can need at least $1/2en - 1/4n^2 - 1/2n + 2$ steps to reach the goal state (for even $n > 2$ and $e \geq 2n-2$)

one dimension with one reflecting and one absorbing barrier [7]. In this case, generating functions can be used to solve Equations 4.

Although the complexity of a random walk decreases in 1-step invertible state spaces, the worst-case complexity of an admissible, zero-initialized Q-learning algorithm with action-penalty representation does *not* change. It remains tight at $O(en)$ in general (an example is given in Figure 19) and $O(n^3)$ for state spaces that have no duplicate actions (an example is shown in Figure 17).

To summarize, the largest average-case complexity of a random walk is polynomial (and no longer exponential) in n if the state space is Eulerian and has no duplicate actions. In fact, the largest big-O average-case complexity of a random walk equals the big-O worst-case complexity of an admissible, zero-initialized Q-learning algorithm, and we can no longer expect an exponential improvement in expected performance when switching from a zero-initialized Q-learning algorithm with goal-reward representation to one of the other algorithms.

One can seriously consider random walks as a solution method for reaching a goal state in Eulerian state spaces, since they can no longer be much less efficient than the other algorithms. Note two small advantages of random walks: To perform a random walk, one does not need any space to store information, whereas an undiscounted, zero-

initialized Q-learning algorithm with (at most) action-penalty representation needs $O(e)$ Q-values with $O(\log_2(d))$ bits each. Also, a random walk reaches a goal state in an *infinite* one- or two-dimensional (but not higher-dimensional) gridworld with probability one [7], whereas the other algorithms cannot guarantee to terminate successfully (without modifications).

However, even for state spaces for which their big-O complexities are identical (such as the one in Figure 17) we expect some improvement in performance, e.g. a linear improvement (i.e. improvement by a constant factor), when switching from goal-reward representation to action-penalty representation, since information about the topology of the state space is only remembered immediately in the latter case.

If the state space is 1-step invertible and the agent knows for every action which other action reverses its effects, then it can use chronological backtracking to reach a goal state. Under the assumption that it has to execute an action at least once before it knows its effect, chronological backtracking achieves a worst-case complexity of $O(e)$, since every action (except for the action that leaves the goal state) is executed exactly once in the worst case. In fact, even if the agent initially does not know which action inverts which other action, there is an algorithm for Eulerian state spaces that executes every action at most twice and thus achieves the same big-O worst-case complexity [6]: "Take unexplored edges whenever possible. If stuck, consider the closed walk of of unexplored edges just completed, and retrace it, stopping at nodes that have unexplored edges, and applying this algorithm recursively from each such node." We call this algorithm the **Deng-Papadimitriou algorithm** after its authors. *No* uninformed on-line search algorithm can do better than $O(e)$, see for example Figure 19, since the agent has to execute every action at least once in the worst case to find out about its effect. However, Q-learning does *not* achieve this complexity. Figure 17 gives an example of a 1-step invertible state space that has $O(n^2)$ actions, but Q-learning may need $O(n^3)$ steps to reach a goal state. Thus, if one considers only Eulerian state spaces, it no longer holds that every uninformed on-line search algorithm has at least the same big-O worst-case complexity as Q-learning. This demonstrates that algorithms that know about special properties of the state space, i.e. that use more knowledge of the state space than Q-learning, can have a smaller big-O worst-case complexity.

4.4.3 Summary

Many reinforcement learning domains have certain properties that decrease the worst-case complexity of uninformed on-line search algorithms. We have focused on gridworlds, because they are popular among reinforcement learning researchers. However, many other search domains that are commonly used in Artificial Intelligence have the properties discussed here. The 8-puzzle, for example, has no identity or duplicate actions, a constant individual upper action bound, and is 1-step invertible.

For Eulerian state spaces, we have shown that the largest big-O average-case complexity of random walks equals the big-O worst-case complexity of admissible, zero-initialized Q-learning algorithms. However, there are uninformed on-line search algorithms that

have a smaller big-O worst-case complexity than Q-learning. This is very different from the general case. There, the largest big-O average-case complexity of random walks is much larger than the big-O worst-case complexity of admissible, zero-initialized Q-learning algorithms, and every uninformed on-line search algorithm has at least the same big-O worst-case complexity as Q-learning. Thus, general results about the behavior of reinforcement learning algorithms might not be specific enough. It can be more illuminating to identify specific properties of the domains of interest and then analyze the behavior of reinforcement learning algorithms in domains that possess these properties.

5 Finding Optimal Policies with Q-Learning

We now consider the problem of finding *shortest* paths from *all* states to a goal state. We present novel extensions of the Q-learning algorithm to solve this problem that have the same big-O worst-case complexity as an admissible, zero-initialized Q-learning algorithm for finding a single arbitrary path from the start state to a goal state. The purpose of these algorithms, that we call **bi-directional Q-learning**, is to be able to explore the state space sufficiently to solve the task and terminate – their purpose is not to exploit the acquired knowledge. First, we describe a version that needs to know an upper bound on the depth of the state space in advance and then proceed to describe a slightly more complicated version that does not need to know any information about the state space in advance.

5.1 Bi-Directional Q-Learning (Version 1)

The algorithm, which we term the **bi-directional** (1-step) **Q-learning algorithm (version 1)**, is presented in Figure 20. It assumes only that the agent can recognize whether it is in a goal state and knows an upper bound $ub(d)$ of d. It could for example know n or an upper bound of n. In the ideal case, it knows the exact value of d. While the complexity results presented here are for the undiscounted, zero-initialized version with action-penalty representation, they can easily be transferred to all of the previously described alternatives.

The bi-directional Q-learning algorithm uses the Q_f-values (U_f-values) to implement an on-line search for a goal state in exactly the same way how the Q-learning algorithm from Figure 3 uses the Q-values (U_f-values).

We say that state $s \in S$ is **done** iff $U_f(s) = U_f^{opt}(s) = -gd(s)$, and that action $a \in A(s)$ is **done** in $s \in S$ iff $Q_f(s,a) = Q_f^{opt}(s,a)$, i.e. $Q_f(s,a) = 0$ for $s \in G$ and $Q_f(s,a) = -1 - gd(succ(s,a))$ for $s \in S \setminus G$. For every $s \in S$, we introduce a value $done(s)$, and, for every $s \in S$ and $a \in A(s)$, we introduce a value $done(s,a)$, with the following semantics: $done(s) = true$ iff the agent knows that $s \in S$ is done, and similarly for $done(s,a)$. Thus, if $done(s) = true$, then $U_f(s) = -gd(s)$ (but not

Initially, $Q_f(s,a) = Q_b(s,a) = 0$ and $done(s,a) = false$ for all $s \in S$ and $a \in A(s)$.
/* Also, no steps have been taken so far, i.e. $t = 0$. */

1. Set $s :=$ the current state.
2. If $s \in G$, then set $done(s,a) := true$ for all $a \in A(s)$.
3. If $done(s) = true$, then go to 8.
4. /* forward step */
   Set $a := \text{argmax}_{a' \in A(s)} Q_f(s,a')$.
5. Execute action a.
   /* As a consequence, the agent receives reward -1 and is in state $succ(s,a)$.
   Increment the number of steps taken, i.e. set $t := t + 1$. */
6. Set $Q_f(s,a) := -1 + U_f(succ(s,a))$ and $done(s,a) := done(succ(s,a))$.
7. Go to 1.
8. /* backward step */
   Set $a := \text{argmax}_{a' \in A(s)} Q_b(s,a')$.
9. Execute action a.
   /* As a consequence, the agent receives reward -1 and is in state $succ(s,a)$.
   Increment the number of steps taken, i.e. set $t := t + 1$. */
10. Set $Q_b(s,a) := -1 + U_b(succ(s,a))$.
11. If $U_b(s) < -ub(d)$, then stop.
12. Go to 1.

where
$U_f(s) := \max_{a \in A(s)} Q_f(s,a)$,
$done(s) := \exists_{a \in A(s)}(Q_f(s,a) = \max_{a' \in A(s)} Q_f(s,a') \wedge done(s,a))$, and
$U_b(s) := \max_{a \in A(s)} Q_b(s,a)$
at every point in time.

Figure 20: The bi-directional Q-learning algorithm (version 1)

necessarily the other way around) and $done(s)$ remains *true* until termination.

Initially, $done(s) = false$ for all $s \in S$, but the agent can gain additional knowledge according to the following three rules:

1. If $s \in G$ and $Q_f(s,a) = 0$, then $a \in A(s)$ is done in $s \in S$.
2. If $done(succ(s,a)) = true$ and $Q_f(s,a) = -1 + U_f(succ(s,a))$, then $a \in A(s)$ is done in $s \in S \setminus G$.
3. If $done(s,a) = true$ and $U_f(s) = Q_f(s,a)$ for an $a \in A(s)$, then $s \in S$ is done (since the Q_f-values are monotonically decreasing).

If the agent was in a state s for which $done(s)$ was *false*, executed action a, and afterwards is in a state s' for which $done(s')$ is *true*, then $done(s,a)$ was *false* before the step, but can be set to *true* afterwards. We call such a transition an **important transition**. After the agent has made at most e important transitions, all n states are done. If the agent is in a state s with $done(s) = false$ and uses Q-learning to reach a goal state, then it will finally make an important transition, since it sets $done(s')$ to *true* in $s' \in G$ if it was not already set to *true*. Immediately after the important transition, the agent can use another Q-learning algorithm with different Q_b-values to reach a state in $G_b := \{s \in S \setminus G : done(s) = false\}$, and will eventually reach such a state if one still exists, in which case it uses the Q_f-values again to reach a goal state. If such a state no longer exists, the U_b-values will decrease without a limit. The algorithm can terminate when a U_b-value drops below $-ub(d)$, since there cannot be shortest paths to any state that are longer than d. Thus, the agent knows that $done(s) = true$ for all $s \in S$, can conclude that $U_f(s) = -gd(s)$ for all $s \in S$, and may terminate. (This termination criterion is peculiar to version 1 of the bi-directional Q-learning algorithm, and will be replaced in version 2.) Then, an optimal policy is to select action $\text{argmax}_{a \in A(s)} U_f(succ(s,a))$ or, equivalently, $\text{argmax}_{a \in A(s), done(s,a) = true} Q_f(s,a)$ in state $s \in S \setminus G$, where $Q_f(s,a)$ and $U_f(s)$ are the Q_f-values and U_f-values upon termination, since it is optimal to always execute an action that decreases the goal distance most (i.e., in this case, by one).

To summarize, the bi-directional Q-learning algorithm iterates over two independent Q-learning searches: a **forward phase** that uses Q_f-values to search for a state s' with $done(s') = true$ from a state s with $done(s) = false$, followed by a **backward phase** that uses Q_b-values to search for a state s' with $done(s') = false$ from a state s with $done(s) = true$. The forward and backward phases are implemented using the Q-learning algorithm from Figure 3. Every forward phase sets at least one additional $done(s,a)$ value to *true* and then transfers control to the backward phase, which continues until a state s with $done(s) = false$ is reached, so that the next forward phase can begin.

Theorem 5 *The bi-directional Q-learning algorithm (version 1) finds an optimal policy and terminates after at most* $O(e \times ub(d))$ *steps.*

The proof of Theorem 5 is similar to that of Theorem 2. The bi-directional Q-learning algorithm can be made more efficient, for example by breaking ties more intelligently, but this does not change its big-O worst-case complexity.

The agent has to make sure that the value of $ub(d)$ that it uses does not underestimate d. If $ub(d)$ scales linearly with the correct value of d, then $O(e \times ub(d)) = O(ed)$. This rules out that the agent uses, for example, $ub(d) \geq d^2$, just to be conservative. Since $d \leq n - 1$ in every strongly connected state space, it holds that $O(ed) \leq O(en)$. If the agent uses $n - 1$ as an upper bound of d, the bi-directional Q-learning algorithm (version 1) has worst-case complexity $O(en)$, but *not* necessarily $O(ed)$, since d can increase sublinearly in n. If the state space has no duplicate actions, the complexity becomes $O(n^3)$. That these bounds are tight follows from Figures 11 and 15, since finding optimal policies cannot be easier than reaching a goal state for the first time. So, if $ub(d)$ scales linearly with d (e.g. because the agent uses $ub(d) = d$), then the bi-directional Q-learning algorithm (version 1) has exactly the same big-O worst-case complexity as the Q-learning algorithm for finding any path from the start state to a goal state. This is surprising, since one could have expected finding optimal policies to be harder for Q-learning than simply reaching a goal state.

5.2 Bi-Directional Q-Learning (Version 2)

The bi-directional Q-learning algorithm that we have presented in the previous chapter needs to know an upper bound $ub(d)$ on the depth of the state space. $ub(d)$ is *only* needed to provide a termination criterion. If the algorithm only has to find an optimal policy in the limit, but does not need to terminate, $ub(d)$ is not needed. Of course, then the agent always explores and never exploits.

It is easy to construct an algorithm that finds an optimal policy and terminates but does not need to know an upper bound on the depth of the state space or any other information about the state space in advance if we do not require the algorithm to be memoryless. Note that the worst-case complexity of any algorithm that finds optimal policies cannot be less than $O(ed)$ (no matter how the algorithm determines that an optimal policy has been found), since finding optimal policies cannot be easier than reaching a goal state for the first time and Figure 11 showed that this has a worst-case complexity of at least $O(ed)$.

In the following, we show a version of the bi-directional Q-learning algorithm that does not require the agent to know anything about the state space in advance, needs an internal memory of at most $O(\log_2 e)$ bits, and has a tight worst-case complexity of $O(ed)$. To distinguish the version described earlier from the one introduced here, we continue to call the original version "bi-directional Q-learning algorithm (version 1)" (or, shorter, "bi-directional Q-learning algorithm") and refer to the new version as "bi-directional Q-learning algorithm (version 2)". Version 2 not only requires no initial knowledge of the state space, but also remedies the problem that the complexity might be large, only because an overly conservative estimate of d is used.

The **bi-directional Q-learning algorithm (version 2)** is shown in Figure 21. It differs from version 1 (shown in Figure 20) only in the termination criterion. To implement the termination criterion, version 2 maintains one additional variable that version 1 does not use. The variable *memory* implements an internal memory that the agent maintains across action executions. *memory* stores integer values that range from 0 to at most e. Thus, the internal memory needs at most $\lceil \log_2(e+1) \rceil$ bits, which is not enough to store a map of the state space.

The general idea behind the termination criterion is as follows. The agent can terminate if $done(s) = true$ for all $s \in S$, since then an optimal policy has been found. Initially, $done(s) = false$ for all $s \in S$. Thus, $done(s) = true$ for all $s \in S$ if $done(s) = true$ for every state s that the agent has already **explored**, i.e. visited at least once, and the agent has explored all states. Every action that the agent has not yet **explored**, i.e. executed at least once, could potentially lead to an unexplored state. Thus, the agent can be sure that it has explored all states iff it has explored all actions. This reasoning leads to the following idea. The agent maintains in its internal memory the sum of the number of unexplored actions that it knows about and the number of states s with $done(s) = false$ that it knows about. It can terminate when this sum reaches zero, since then $done(s) = true$ for all explored states $s \in S$ and all states have been explored, i.e. $done(s) = true$ for all $s \in S$.

This idea could be implemented exactly as stated. However, a couple of additional observations simplify the implementation. For example, note that all actions in a non-goal state s have been explored if $done(s) = true$. Therefore, the algorithm does not need to keep track of unexplored actions in non-goal states. Similarly, $done(s) = true$ for a goal state s if all actions in s have been explored. Thus, the algorithm does not need to keep track of the number of goal states s with $done(s) = false$. In short, the agent needs to maintain in its internal memory (i.e. the variable *memory*) only the sum of the number of unexplored actions *in goal states* that it knows about and the number of *non-goal* states s with $done(s) = false$ that it knows about. Again, the agent knows that it can terminate when this sum reaches zero.

This termination criterion can easily be implemented. Initially, $memory = 0$.

- Whenever the agent is in a state s that was still unexplored one step ago, then this state has not been taken into account when calculating the sum, and the sum has to be adjusted properly. If s is a goal state, the sum has to be increased by $|A(s)|$, since all of its $|A(s)|$ actions are still unexplored. If s is not a goal state, *memory* has to be increased by one, since $done(s)$ was *false* initially and remains *false* for at least as long as no action has been executed in s.

- Whenever $done(s)$ changes for a non-goal state s, the sum has to be adjusted properly. $done(s)$ can only change for the current state, it can only change from *false* to *true*, and it can only change after an action execution in the forward step. Furthermore, a forward step is only executed for non-goal states. Thus, when $done(s)$ changes for the current state s after action execution in a forward step, *memory* has to be decreased by one.

Initially, $memory = 0$, $Q_f(s,a) = Q_b(s,a) = 0$, and $done(s,a) = false$ for all $s \in S$ and $a \in A(s)$. /* Also, no steps have been taken so far, i.e. $t = 0$. */

1. Set $s :=$ the current state.
2. If $s \in G$, then set $done(s,a) := true$ for all $a \in A(s)$.
3. If $Q_f(s,a) = Q_b(s,a) = 0$ for all $a \in A(s)$,
   then set $memory := memory +$ (if $s \in G$ then $|A(s)|$ else 1).
4. If $memory = 0$, then stop.
5. If $done(s) = true$, then go to 11.
6. /* forward step */
   Set $a := \mathrm{argmax}_{a' \in A(s)} Q_f(s,a')$.
7. Execute action a.
   /* As a consequence, the agent receives reward -1 and is in state $succ(s,a)$.
   Increment the number of steps taken, i.e. set $t := t + 1$. */
8. Set $Q_f(s,a) := -1 + U_f(succ(s,a))$ and $done(s,a) := done(succ(s,a))$.
9. If $done(s)$ has changed during the execution of step 8,
   then set $memory := memory - 1$.
10. Go to 1.
11. /* backward step */
    Set $a := \mathrm{argmax}_{a' \in A(s)} Q_b(s,a')$.
12. Execute action a.
    /* As a consequence, the agent receives reward -1 and is in state $succ(s,a)$.
    Increment the number of steps taken, i.e. set $t := t + 1$. */
13. Set $Q_b(s,a) := -1 + U_b(succ(s,a))$.
14. If $s \in G$ and $Q_b(s,a)$ has changed from zero to non-zero during the execution of step 13, then set $memory := memory - 1$.
15. Go to 1.

where
$U_f(s) := \max_{a \in A(s)} Q_f(s,a)$,
$done(s) := \exists_{a \in A(s)} (Q_f(s,a) = \max_{a' \in A(s)} Q_f(s,a') \wedge done(s,a))$, and
$U_b(s) := \max_{a \in A(s)} Q_b(s,a)$
at every point in time.

Figure 21: The bi-directional Q-learning algorithm (version 2)

- Whenever an action in a goal state becomes explored, the sum has to be adjusted properly. Actions in goal states can only be executed in backward steps. Therefore, when an action in a goal state becomes explored after an action execution in a backward step, *memory* has to be decreased by one.

Thus, the agent needs to be able to detect whether its current state was unexplored one step ago and whether an action in a goal state is unexplored or not. In general, an action $a \in A(s)$ has never been executed in a forward step iff $Q_f(s,a) = 0$. Similarly, it has never been executed in a backward step iff $Q_b(s,a) = 0$. Thus, the current state s of the agent was unexplored one step ago iff the agent has never executed any action in s, i.e. $Q_b(s,a) = Q_f(s,a) = 0$ for all $a \in A(s)$. An action $a \in A(s)$ in a *goal state* s is unexplored iff $Q_b(s,a) = 0$, since actions in goal states can only be executed in backward steps and therefore it always holds that $Q_f(s,a) = 0$.

These thoughts translate directly to the bi-directional Q-learning algorithm (version 2). Since version 1 and version 2 of the bi-directional Q-learning algorithm differ only in the termination criterion, both behave identically after every step and therefore find the same optimal policy (if ties are broken in the same way). Thus, it is still optimal to select action $\text{argmax}_{a \in A(s)} U_f(succ(s,a))$ or, equivalently, $\text{argmax}_{a \in A(s), done(s,a)=true} Q_f(s,a)$ in state $s \in S \setminus G$, where $Q_f(s,a)$ and $U_f(s)$ are the Q_f-values and U_f-values upon termination.

Theorem 6 *The bi-directional Q-learning algorithm (version 2) finds an optimal policy and terminates after at most $O(ed)$ steps.*

To prove this theorem, one shows that the termination criterion of the bi-directional Q-learning algorithm (version 2) is implied by the one of the bi-directional Q-learning algorithm (version 1). This shows that version 2 always terminates no later than version 1 no matter what $ub(d)$ is (provided that ties are broken in the same way).[15] If $ub(d) = d$, then the worst-case complexity of the bi-directional Q-learning algorithm (version 1) is $O(ed)$. Therefore, the worst-case complexity of version 2 is $O(ed)$ as well. If the state space has no duplicate actions, the complexity becomes $O(n^3)$. That these bounds are tight follows from Figures 11 and 15, since finding optimal policies cannot be easier than reaching a goal state for the first time.

5.3 Comparison of Bi-Directional Q-Learning to other On-Line Search Algorithms

To find optimal policies, the agent has to execute every action at least once in order to find out about its effect. If the state space is Eulerian, then the Deng-Papadimitriou

[15] To be precise: The bi-directional Q-learning algorithm (version 2) checks the termination criterion a couple of lines later than version 1. Thus, version 2 always terminates at most a couple of program statements later than version 1.

algorithm can be used to learn a map which is then used for finding optimal policies. It executes every action at most twice, i.e. it has complexity $O(e)$ and is competitive. *No* uninformed on-line search algorithm can do better. However, the bi-directional Q-learning algorithms do *not* achieve this complexity, as shown in Figure 17. This shows again that on-line search algorithms that know about special properties of the state space can have an advantage over less informed algorithms. First worst-case results for uninformed on-line search algorithms in non-Eulerian state spaces are reported in [6].

6 Complexity of Value-Iteration

The results derived for Q-learning can easily be transferred to value-iteration. In the following, we state the main definitions, lemmas, and theorems as they apply to value-iteration.

Definition 12 *A value-iteration algorithm is* **initialized** *with $q \in \mathcal{R}$ (or, synonymously, q-initialized) iff initially, for all $s \in G$, $U(s) = 0$, and, for all $s \in S \setminus G$, $U(s) = q$.*

6.1 Reaching a Goal State with Value-Iteration

A zero-initialized value-iteration algorithm with goal-reward representation performs a random walk. Therefore, Theorem 7 follows immediately from Theorem 1, and Theorem 8 follows from Theorem 4.

Theorem 7 *The expected number of steps that a zero-initialized value-iteration algorithm with goal-reward representation needs in order to reach a goal state and terminate can be exponential in n.*

Theorem 8 *An zero-initialized value-iteration algorithm with goal-reward representation reaches a goals state and terminates after at most $O(ed)$ steps on average if the state space is Eulerian.*

An undiscounted value-iteration algorithm that always executes the action that leads to the state with the largest U-value is equivalent to the **Learning Real-Time A*** (LRTA*) **algorithm** [15] [16] [17] with a search horizon of one if the state space is deterministic and the action penalty representation is used [2]. ([2] also showed that the LRTA* algorithm can be generalized to probabilistic domains.) Ishida and Korf proved that a zero-initialized LRTA* algorithm is guaranteed to reach a goal state in at most $O(n^2)$ steps if the state space has no identity actions. This follows from the analysis of the Moving Target Search (MTS) algorithm in [10] if the position of the target does not change, see [14] for a detailed discussion and bibliography. As

argued earlier, the agent does not need to execute identity actions, since they cannot be part of an optimal plan. However, the value-iteration algorithm treats an identity action exactly in the same way that it treats every other action (although it could be augmented to identify and avoid identity actions). Thus, the presence of identity actions can potentially make the path planning task harder.

The following theorems about value-iteration generalize Ishida and Korf's result to state spaces that can contain identity actions and show the effects of various domain properties and initial U-values that are non-zero on the worst-case complexity. Definition 13, 14, and 15, as well as Lemma 3 assume no discounting, but can easily be modified for the discounted case.

Definition 13 *U-values are* **consistent** *iff, for all $s \in G$, $U(s) = 0$, and, for all $s \in S \setminus G$, $\max_{a \in A(s)}(-1 + U(succ(s,a))) \leq U(s) \leq 0$.*

Definition 14 *U-values are* **admissible** *iff, for all $s \in S$, $-gd(s) \leq U(s) \leq 0$.*

Zero-initialized U-values are consistent, and consistent U-values are admissible. U-values are admissible iff $-U$ is an admissible heuristic for the goal distances of the states in the context of A*-search.

Definition 15 *A value-iteration algorithm is* **admissible** *iff it uses action-penalty representation, its action selection step is "$a := argmax_{a' \in A(s)}(\overline{r}(s,a') + U(succ(s,a'))) = argmax_{a' \in A(s)}(-1 + U(succ(s,a')))$,"[16] and either*

- *its initial U-values are consistent, and its value update step[17] is "Set $U(s) := -1 + U(succ(s,a))$," or*
- *its initial U-values are admissible, and its value update step is "Set $U(s) := \min(U(s), -1 + U(succ(s,a)))$."*

If a value-iteration algorithm is admissible, then consistent (admissible) U-values remain consistent (admissible) after every step of the agent and are monotonically decreasing. A time superscript of t in the following lemma refers to the values of the variables immediately before the action execution step, i.e. line 4, of step t.

Lemma 3 *For all steps $t \in \mathcal{N}_0$ (until termination) of an undiscounted, admissible value-iteration algorithm,*

$$U^t(s^t) + \sum_{s \in S} U^0(s) - t \geq \sum_{s \in S} U^t(s) + U^0(s^0) - loop^t$$

[16] Since $\overline{r}(s,a') = -1$ for all $s \in S \setminus G$ and $a' \in A(s)$, the action selection step can be simplified to "$a := \text{argmax}_{a' \in A(s)} U(succ(s,a'))$."

[17] Note that the action selection step "$a := \text{argmax}_{a' \in A(s)}(-1 + U(succ(s,a')))$" allows us to simplify the value update step from the general "Set $U(s) := \max_{a' \in \overline{A}(s)}(\overline{r}(s,a') + \gamma U(\overline{succ}(s,a')))$.", that we used in the statement of the value-iteration algorithm in Figure 4, to the more specific "Set $U(s) := \overline{r}(s,a) + \gamma U(succ(s,a)) = -1 + \gamma U(succ(s,a)) = -1 + U(succ(s,a))$".

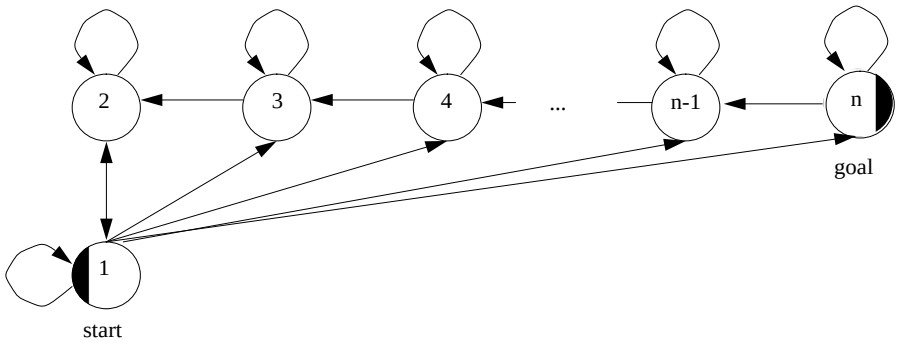

Figure 22: A domain for which an admissible, zero-initialized value-iteration algorithm can need at least $n^2 - n$ steps to reach the goal state, but an off-line algorithm that knows the topology of the state space needs only one step to reach the goal state (for $n \geq 1$)

and

$$loop^t \leq \sum_{s \in S} U^0(s) - \sum_{s \in S} U^t(s)$$

where $loop^t := |\{t' \in \{0, \ldots, t-1\} : s^{t'} = s^{t'+1}\}|$ *(the number of identity actions executed before* t*).*

All results about undiscounted value-iteration algorithms can be transferred to the discounted case, as outlined for Q-learning.

Lemma 4 *An admissible value-iteration algorithm reaches a goal state and terminates after at most* $2 \sum_{s \in S} gd(s) \leq n^2 - n$ *steps. If the state space has no identity actions, it reaches a goal state and terminates after at most* $\sum_{s \in S} gd(s) \leq 0.5n^2 - 0.5n$ *steps.*

Both bounds are tight for zero-initialized value-iteration, as shown in Figures 22 and 23. (It is important to keep in mind that the agent can only detect whether its current state is a goal state. Therefore, it cannot detect that state n is a goal state when it is in state 1.) Note that the worst-case complexity doubles if identity actions are present. However, the big-O worst-case complexity is the same in both cases.

Theorem 9 *An admissible value-iteration algorithm reaches a goal state and terminates after at most* $O(nd)$ *steps.*

This theorem applies to admissible value-iteration algorithms that are zero-initialized. Note that the worst-case complexity does *not* depend on e, whereas the analogous complexity for Q-learning does, see Theorem 2. This is so, because value-iteration always prefers duplicate actions equally, since duplicate actions have the same outcome

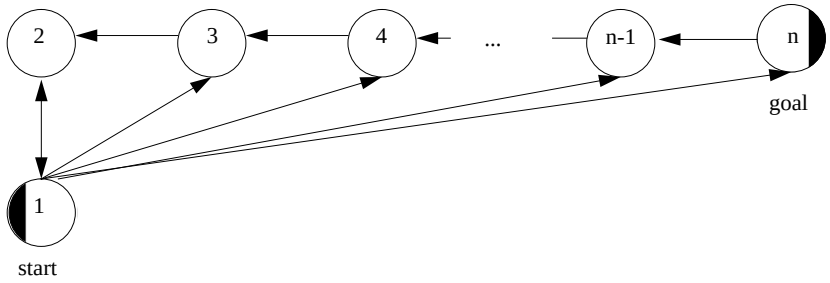

Figure 23: A domain for which an admissible, zero-initialized value-iteration algorithm (and every other algorithm that has to enter a state at least once before it knows the successor states) can need at least $1/2n^2 - 1/2n$ steps to reach the goal state, but an off-line algorithm that knows the topology of the state space needs only one step to reach the goal state (for $n \geq 1$)

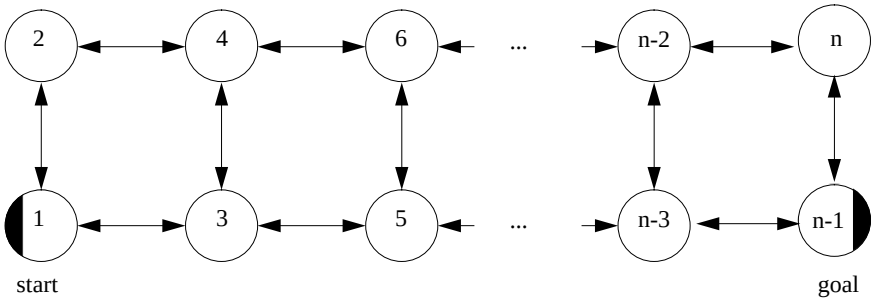

Figure 24: A domain for which an admissible, zero-initialized value-iteration algorithm can need at least $3/16n^2 - 3/4$ steps to reach the goal state (for $n \geq 1$ with $n \bmod 4 = 2$)

and actions are evaluated according to the U-values of their outcomes. Thus, value-iteration does *not* necessarily need to execute every action at least once. Q-learning, in contrast, evaluates actions according to their Q-values, and the Q-values of duplicate actions can differ.

$O(nd) \leq O(n^2)$, since $d \leq n - 1$. The worst-case complexity of $O(n^2)$ is tight even if the state space has no duplicate actions, a constant individual upper action bound, *and* is 1-step invertible, see the rectangular gridworld in Figure 24.

Every algorithm that has to enter a state at least once before it knows the successor states of that state has a worst-case complexity of at least $O(n^2)$, see Figure 23. This holds even if the state space has no duplicate actions *and* a constant individual upper action bound, see Figure 25, and thus these algorithms are *not* competitive. This includes algorithms that learn a map of the state space. However, if the state space

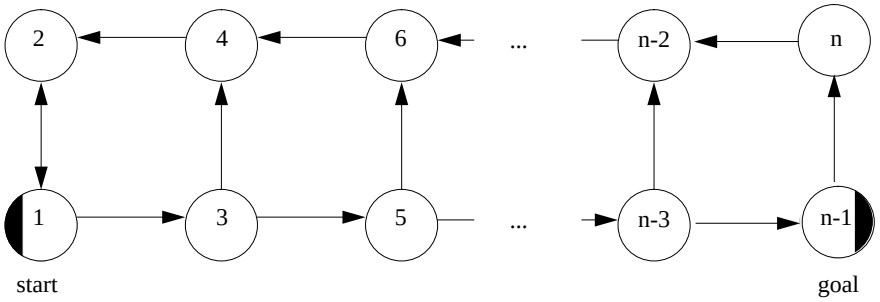

Figure 25: A domain for which an admissible, zero-initialized value-iteration algorithm (and every other algorithm that has to enter a state at least once before it knows the successor states) can need at least $1/4n^2 - 1$ steps to reach the goal state (for even $n > 1$)

is 1-step invertible, then the agent can identify which action inverts the action that it executed last and thus use chronological backtracking to reach a goal state. This decreases the worst-case complexity to $O(n)$, since the agent leaves every state at most once. *No* algorithm can do better in the worst case, see for example Figure 18. However, value-iteration does *not* achieve this complexity, as shown above with Figure 24.

Similarly to Q-learning, it holds that a discounted, one-initialized value-iteration algorithm with goal-reward representation behaves exactly like an undiscounted, (minus one)-initialized value-iteration algorithm with action-penalty representation if ties are broken in the same way.

Theorem 10 *A discounted, one-initialized value-iteration algorithm with goal-reward representation reaches a goal state and terminates after at most $O(nd)$ steps if its action selection step is "$a := argmax_{a' \in A(s)}(\overline{r}(s,a') + U(succ(s,a')))$." and its value update step is "Set $U(s) := \overline{r}(s,a) + \gamma U(succ(s,a))$."*

All of the above complexity results can be utilized to prove worst-case complexities for Q-learning, since value-iteration behaves like Q-learning in a slightly modified state space. Assume that Q-learning operates in a domain with states S, start state s_{start}, goal states G, actions $A(s)$ for $s \in S$, successor function $succ$, and initial Q-values $Q(s,a)$ for $s \in S$ and $a \in A(s)$. We refer to this domain in the following as the original domain. Consider the following transformed domain:

$$
\begin{array}{rcll}
\widetilde{S} & := & \{s_{s,a} : s \in S, a \in A(s)\} & \\
\widetilde{s}_{start} & := & s_{s_{start}, argmax_{a \in A(s_{start})} Q(s_{start},a)} & \\
\widetilde{G} & := & \{s_{s,a} : s \in G, a \in A(s)\} & \\
\widetilde{A}(\widetilde{s}) & := & A(succ(s,a)) & \text{for all } \widetilde{s} = s_{s,a} \in \widetilde{S} \\
\widetilde{succ}(\widetilde{s},\widetilde{a}) & := & s_{succ(s,a),\widetilde{a}} & \text{for all } \widetilde{s} = s_{s,a} \in \widetilde{S} \text{ and } \widetilde{a} \in \widetilde{A}(\widetilde{s})
\end{array}
$$

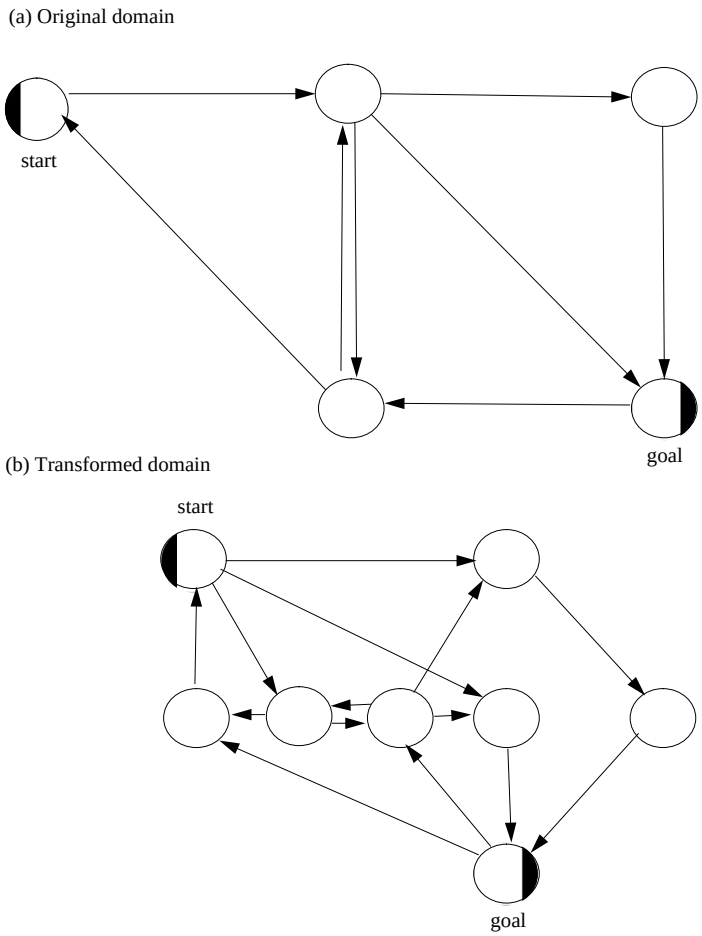

Figure 26: A domain and its transformation

where the $s_{s,a}$ are new states. Basically, the actions of the original domain become the the states of the transformed one. The potential successor states of such an "action state" $s_{s,a}$ are the "action states" that correspond to the actions that can be executed immediately after the execution of a in s, i.e. the set $A(succ(s,a))$. Note that the transformed domain satisfies all properties that we require in this report to hold for state spaces. In particular, it is strongly connected. Note also that the size of the transformed domain $\tilde{n}$ equals the total number of actions e of the original domain. Similarly, the depth $\tilde{d}$ of the transformed domain is either equal to either the depth of the original domain d or to $d+1$. An example for an original domain and the transformed domain that corresponds to it is given in Figure 26.

Assume that action penalty representation and no discounting are used. Admissible value-iteration behaves in the transformed domain identically to admissible Q-learning in the original one provided that

- the original domain has no identity actions,
- initially $Q(s,a) = U(s_{s,a})$ for all $s \in S$ and $a \in A(s)$, and
- ties are broken in the same way.

During the execution of the algorithms it always holds that $Q(s,a) = U(s_{s,a})$ for all $s \in S$ and $a \in A(s)$. Furthermore, value-iteration always chooses the same actions as Q-learning, and needs the same number of steps to terminate. Note that the Q-values of the original domain are consistent (admissible) iff the U-values of the transformed domain are consistent (admissible).

The similarity in the behavior of value-iteration and Q-learning can be used to transfer theorems about one algorithm to the other one. For example, Theorem 9 states that value-iteration has a worst-case complexity of $O(\tilde{n}\tilde{d}) = O(ed)$ in the transformed domain. Thus, this is the worst-case complexity of Q-learning in the original domain as well, which confirms Theorem 2. (As Theorem 2 shows, the conditions that guarantee equal behavior of value-iteration and Q-learning can be generalized to discounted algorithms and state spaces that contain identity actions.)

6.2 Finding Optimal Policies with Value-Iteration

Korf showed that the LRTA* algorithm identifies an optimal path from a given start state to a set of goal states in the limit if the agent is automatically reset to the start state when it reaches a goal state. Our **bi-directional** (1-step) **value-iteration algorithm**, see Figure 27, is more general than Korf's algorithm in that it finds shortest paths from *all* states to a goal state, does *not* need to have reset actions available, and terminates. $ub(d)$ is again an upper bound on the depth of the state space d.[18] See [14] for details on the bi-directional value-iteration algorithm (and how it can be made more efficient, unfortunately without decreasing its big-O worst-case complexity).

Theorem 11 *The bi-directional value-iteration algorithm finds an optimal policy and terminates after at most $O(n \times ub(d))$ steps.*

If $ub(d)$ scales linearly with the correct value of d, then $O(n \times ub(d)) = O(nd) \leq O(n^2)$. That this bound is tight even if the state space has no duplicate actions, a constant

[18] The bi-directional value-iteration algorithm corresponds to the bi-directional Q-learning algorithm (version 1). As in the case of that Q-learning algorithm, the upper bound on the depth of the state space $ub(d)$ is needed only to provide a termination criterion for the bi-directional value-iteration algorithm. If we do not require the agent to be memoryless, then it is easy to construct a version of the bi-directional value-iteration algorithm that does not need to know an upper bound on the depth of the state space or any other information about the state space in advance and has a tight worst-case complexity of $O(nd)$. This can be done along the lines outlined in Chapter 5.2 for the bi-directional Q-learning algorithm (version 2).

Initially, $U_f(s) = U_b(s) = 0$ and $done(s) = false$ for all $s \in S$.
/* Also, no steps have been taken so far, i.e. $t = 0$. */

1. Set $s :=$ the current state.
2. If $s \in G$, then set $done(s) := true$.
3. If $done(s) = true$, then go to 8.
4. /* forward step */
   Set $a := \text{argmax}_{a' \in A(s)} U_f(succ(s, a'))$.
5. Execute action a.
   /* As a consequence, the agent receives reward -1 and is in state $succ(s, a)$. Set $t := t + 1$. */
6. Set $U_f(s) := -1 + U_f(succ(s, a))$ and $done(s) := done(succ(s, a))$.
7. Go to 1.
8. /* backward step */
   Set $a := \text{argmax}_{a' \in A(s)} U_b(succ(s, a'))$.
9. Execute action a.
   /* As a consequence, the agent receives reward -1 and is in state $succ(s, a)$. Set $t := t + 1$. */
10. Set $U_b(s) := -1 + U_b(succ(s, a))$.
11. If $U_b(s) < -ub(d)$, then stop.
12. Go to 1.

Figure 27: The bi-directional value-iteration algorithm

individual upper action bound, *and* is 1-step invertible, follows from Figure 24, since finding optimal policies cannot be easier than reaching a goal state.

Every algorithm that has to enter a state at least once before it knows the successor states of that state has a worst-case complexity of at least $O(n^2)$ for finding an optimal policy even if the state space has no duplicate actions *and* a constant individual upper action bound, see for example Figure 25. However, if one considers only Eulerian state spaces, then the Deng-Papadimitriou algorithm, that has a worst-case complexity of only $O(e)$, can be used to learn a map that is then used for finding an optimal policy. Thus, this algorithm needs at most $O(n)$ steps for finding optimal policies in Eulerian state spaces with a linear total upper action bound, whereas the bi-directional value-iteration algorithm can need $O(n^2)$ steps, as shown above with Figure 24. Similarly, if one considers only 1-step invertible state spaces, chronological backtracking has a complexity of $O(n)$, but bi-directional value-iteration can still need $O(n^2)$ steps, as shown with Figure 24 as well.

7 Empirical Results

Until now, we have been concerned with the worst-case complexity of reinforcement learning algorithms. However, for practical purposes their average-case complexities are equally important. In this chapter, we present a brief case study of the behavior of various uninformed reinforcement learning algorithms in three simple domains:

- reset state spaces (see Figure 8) of sizes $n \in [2, 50]$,
- one-dimensional gridworlds (see Figure 18) of sizes $n \in [2, 50]$, and
- two-dimensional, quadratic gridworlds without obstacles (see Figure 16 for an example with obstacles) of sizes $n \in [4, 196]$ that have the start state in the upper left-hand corner and the goal state in the lower right-hand corner.

The one-dimensional and quadratic gridworlds are 1-step invertible, but the reset state space is not. The depth of the one-dimensional gridworld and the reset state space scales linearly with n, since $d = n - 1$ in both cases. For the quadratic gridworld, however, d scales sublinearly with n, because $d = 2\sqrt{n} - 2$. All three domains have no duplicate actions, a constant individual upper action bound, and polynomial width.

The run-times of the reinforcement learning algorithms (i.e. the number of steps needed) are shown in Figures 28, 29, and 30, respectively. All graphs are scaled in the same proportion.

The x-axis shows the complexity of the domain (measured as ed) and the y-axis the run-time (measured as number of steps needed to complete the tasks). The expected run-time of a random walk (i.e. of a zero-initialized Q-learning or value-iteration algorithm with goal-reward representation) is determined analytically. The run-times

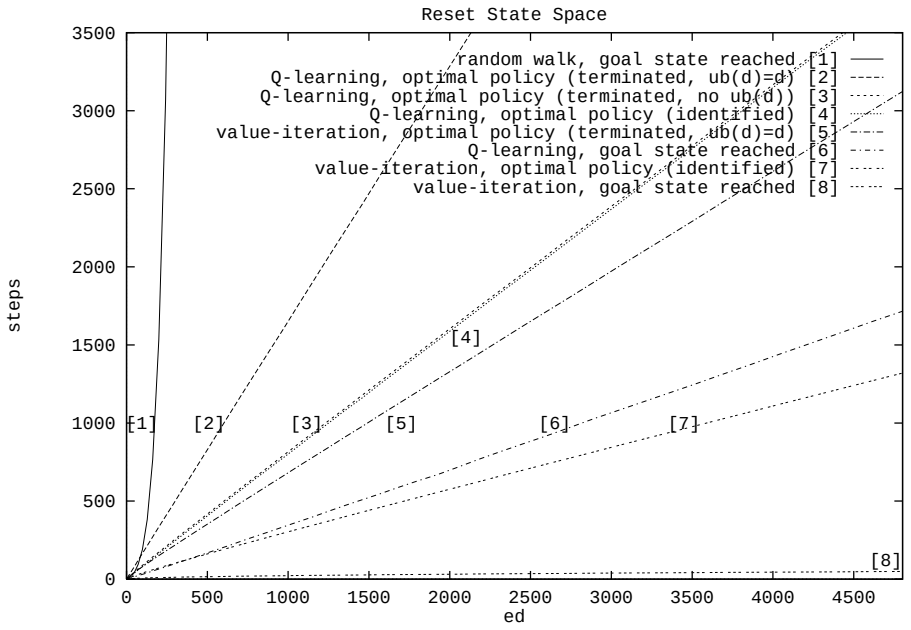

Figure 28: Run-times in the reset state space (as a function of $ed = e(n-1)$)

of the other reinforcement learning algorithms (we use zero-initialized algorithms with action-penalty representation) are averaged over 5000 runs, with ties broken randomly. For finding optimal policies, we use either the bi-directional value-iteration algorithm, the bi-directional Q-learning algorithm (version 1), or its version 2. We label the last case with "no $ub(d)$", since the bi-directional Q-learning algorithm (version 2) does not need to know an upper bound on the depth of the state space in advance.[19] In the first two cases, we use either $ub(d) = d$ or $ub(d) = n-1$ as upper bounds on the depth of the state space. In all three cases, we distinguish two performance measures: the number of steps until $U_f(s) = U^{opt}(s)$ for every $s \in S$ (i.e. until an optimal policy is identified), and the number of steps until the algorithm realizes that and terminates. For identifying an optimal policy, it does not matter which version of the bi-directional Q-learning algorithm is used.

Note that value-iteration needs exactly $n-1$ steps to reach the goal state in reset state spaces and one-dimensional gridworlds, the best number of steps possible. These graphs are thus very close to the x-axis. Also, the bi-directional Q-learning algorithm (version 2) terminates (almost) immediately after an optimal policy has been identified. Thus, the corresponding graphs are very close.

Note that d and $n-1$ are identical for the reset state space and the one-dimensional gridworld, but they differ for the quadratic gridworld. Thus, $ub(d) = d$ and $ub(d) = n-1$ collapse for the former two domains and, furthermore, it does not matter whether

[19] We do not include the results about the corresponding version of the bi-directional value-iteration algorithm, since we have not described it in this report.

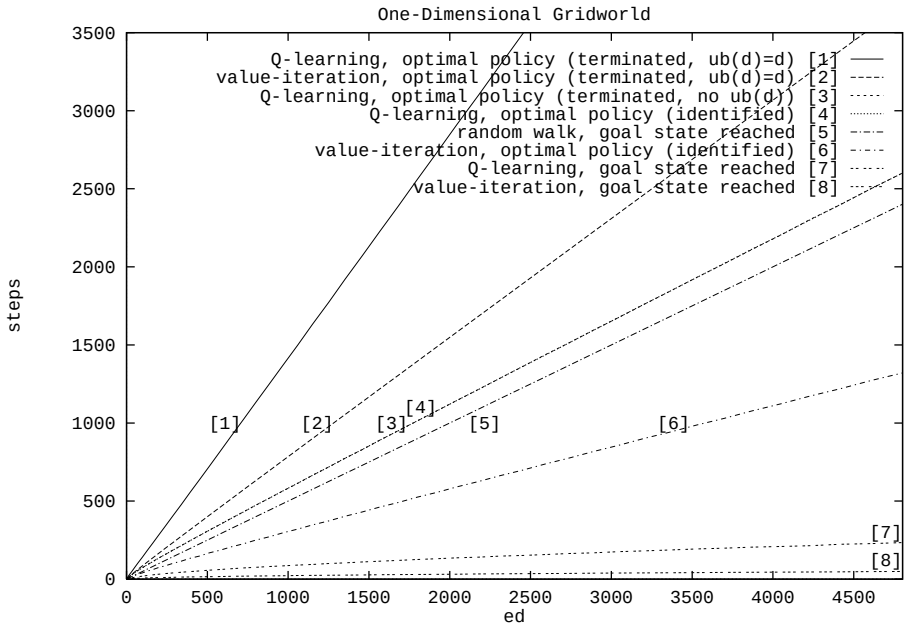

Figure 29: Run-times in the one-dimensional gridworld (as a function of $ed = e(n-1)$)

we use ed or en as a measure for the complexity of the domain. For the latter domain, however, d scales sublinearly with n. Thus, it does make a difference whether we use $ub(d) = d$ or $ub(d) = n - 1$ and whether we use ed or en as a measure for the domain complexity. Figure 31 contains the same data as Figure 30, except that it uses en for the domain complexity instead of ed.

In the following, we investigate how the run-times of the algorithms scale with the complexity of the domains. We have shown in the previous chapters that the worst-case bounds of the following tasks can be at most linear in ed:

- reaching a goal state with random walks in Eulerian state spaces on average,
- reaching a goal state with (efficient) Q-learning (i.e. using either zero-initialized Q-learning with action-penalty representation or one-initialized Q-learning with goal-reward representation),
- identifying an optimal policy with (efficient) Q-learning, and
- identifying an optimal policy and terminating with (efficient) Q-learning if either version 1 is used and $ub(d)$ scales at most linearly with d or version 2 is used.

Since $d \leq n - 1$, these bounds are also at most linear in en. If d scales sublinearly with n, they are even sublinear in en.

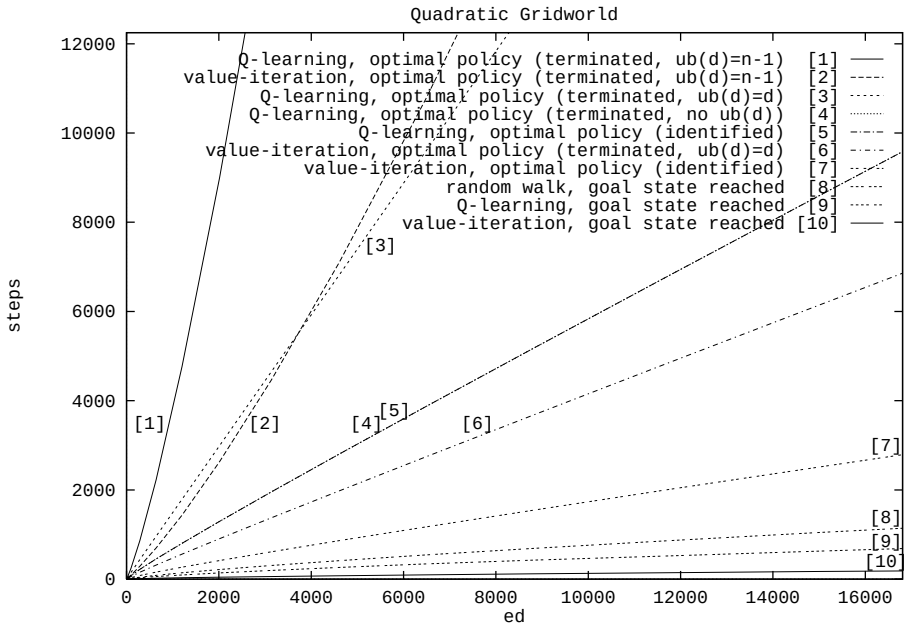

Figure 30: Run-times in the quadratic gridworld (as a function of ed)

Remember that the worst-case complexity for reaching a goal state with efficient Q-learning depends on $\sum_{s \in S \setminus G} \sum_{a \in A(s)} (gd(succ(s,a))+1)$ (see Lemma 2), of which $e(d+1)$ and en are upper bounds. Table 1 contains these values for the three domains.

domain	$\sum_{s \in S \setminus G} \sum_{a \in A(s)} (gd(succ(s,a))+1)$	$e(d+1)$	en
reset state space	$1.5n^2 - 2.5n$	$2n^2 - 2n$	$2n^2 - 2n$
one-dimensional gridworld	$n^2 - 3$	$2n^2 - 2n$	$2n^2 - 2n$
quadratic gridworld	$4n^{3/2} - 4n - 4$	$8n^{3/2} - 12n + 4n^{1/2}$	$4n^2 - 4n^{3/2}$

Table 1: Domain characteristics

Thus,

$$O(\sum_{s \in S \setminus G} \sum_{a \in A(s)} (gd(succ(s,a))+1)) = O(ed) = O(en)$$

for reset state spaces and one-dimensional gridworlds, but

$$O(\sum_{s \in S \setminus G} \sum_{a \in A(s)} (gd(succ(s,a))+1)) = O(ed) \ll O(en)$$

for quadratic gridworlds, since d is linearly proportional to n for the former two domains, but sublinearly proportional for the latter one. In all three cases, ed is linearly proportional to $\sum_{s \in S \setminus G} \sum_{a \in A(s)} (gd(succ(s,a))+1)$ and therefore a good measure for

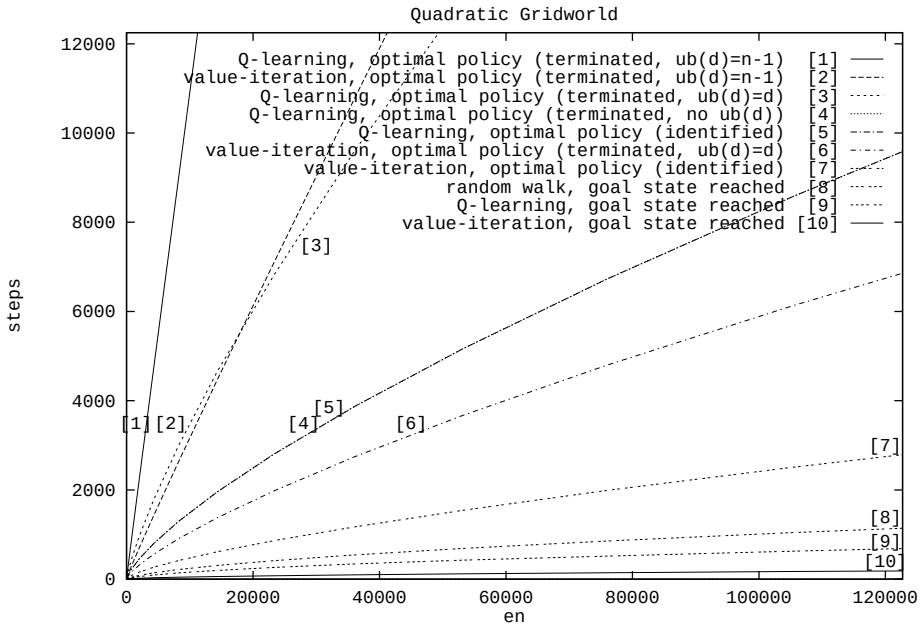

Figure 31: Run-times in the quadratic gridworld (as a function of en)

the worst-case complexity of reaching a goal state with efficient Q-learning. Although this does not need to be the case, it seems to hold for almost all domains of practical interest.

The worst-case bound of the following task can be at most linear in en:

- identifying an optimal policy and terminating with (efficient) Q-learning if version 1 is used and $ub(d)$ scales at most linearly with n.

The worst-case bounds of the following tasks can be at most linear in nd:

- reaching a goal state with (efficient) value-iteration,
- identifying an optimal policy with (efficient) value-iteration, and
- identifying an optimal policy and terminating with (efficient) value-iteration if $ub(d)$ scales at most linearly with d.

Since $n \leq e$, these bounds are also at most linear in ed. They can be exactly linear in ed if the state space has a linear total upper action bound. The bounds are at most linear in n^2, because $d \leq n-1$. If d scales sublinearly with n, they are sublinear in n^2.

The worst-case bound of the following task can be at most linear in n^2:

- identifying an optimal policy and terminating with (efficient) value-iteration if $ub(d)$ scales at most linearly with n.

Since $n \leq e$, the bound is also at most linear in en. It can be exactly linear in en if the state space has a linear total upper action bound.

Although the worst-case complexity of an algorithm provides an upper bound for its average-case complexity, the average-case complexity does *not* necessarily scale linearly with the worst-case complexity. Therefore, it is interesting to investigate in which of the cases not only the worst-case complexity, but also the average-case complexity scales linearly with the complexity of the domain.

Table 2 shows which graphs of the average-case complexity deviate from the corresponding graphs of the worst-case complexity. The first entry is always the shape of the graph for the worst-case complexity (as stated above, i.e. taking into account only the general properties of the state space such as being 1-step invertible etc.), the second entry is the one of the graph for the average-case complexity. We call a graph **superlinear** if it has a positive second derivative, **linear** if its second derivative is zero, and **sublinear** if its second derivative is negative.[20]

This table demonstrates that – at least for the domains tested – the worst-case complexity of the algorithms for finding optimal policies behaves similar to their average-case complexity. Note, however, that the coefficients for an algorithm (i.e. the slopes of the graphs) are domain dependent.

Figure 32 shows the complexity (measured as ed) and Figure 33 the sizes (measured as n) of domains for which the algorithms require 1000 steps on average. We use a simple linear approximation scheme between data points for cardinal n. (Reaching goal states with value-iteration needs so few steps that its graph is far above the other graphs and therefore not contained in the figures.) These graphs confirm our expectations about the algorithms:

- We expect the run-time for reaching a goal state to be smaller than the run-time for finding an optimal policy which we expect, in turn, to be smaller than the run-time for terminating with an optimal policy, given that the same algorithm, task representation, and initialization is used in all cases: Reaching a goal state is a prerequisite for finding an optimal policy, which is necessary for terminating with an optimal policy.

- We also expect a Q-learning algorithm with "no $ub(d)$" to terminate with an optimal policy earlier than a Q-learning algorithm with $ub(d) = ub_1$, which we

[20] The shapes of the graphs for the average-case complexity were determined by visual inspection of the graphs only. Therefore, there is for example a chance that a non-linear shape appeared linear, or that a sublinear or superlinear shape was the result of lower order terms. For example, the graph $(x, f(x)) = (n^2, n^2 + n)$ for $n \in [1, 10]$ is sublinear, although $n^2 \leq n^2 + n \leq 2n^2$ and therefore $O(n^2) = O(n^2 + n)$. Of course, for large values of the independent variable n, the graph approaches a linear shape. Unfortunately, we cannot be sure that in our experiments n was large enough.

domain	reset state space		one-dimensional gridworld	
domain complexity	ed or en		ed or en	
	worst case	average case	worst case	average case
random walk, goal state reached	superlinear	superlinear	linear	linear
value-iteration, goal state reached	linear	sublinear	linear	sublinear
value-iteration, optimal policy (identified)	linear	linear	linear	linear
value-iteration, optimal policy (terminated, $ub(d) = d$)	linear	linear	linear	linear
value-iteration, optimal policy (terminated, $ub(d) = n - 1$)	linear	linear	linear	linear
Q-learning, goal state reached	linear	linear	linear	sublinear
Q-learning, optimal policy (identified)	linear	linear	linear	linear
Q-learning, optimal policy (terminated, no $ub(d)$)	linear	linear	linear	linear
Q-learning, optimal policy (terminated, $ub(d) = d$)	linear	linear	linear	linear
Q-learning, optimal policy (terminated, $ub(d) = n - 1$)	linear	linear	linear	linear

domain	quadratic gridworld			
domain complexity	ed		en	
	worst case	average case	worst case	average case
random walk, goal state reached	linear	sublinear	sublinear	sublinear
value-iteration, goal state reached	linear	sublinear	sublinear	sublinear
value-iteration, optimal policy (identified)	linear	linear	sublinear	sublinear
value-iteration, optimal policy (terminated, $ub(d) = d$)	linear	linear	sublinear	sublinear
value-iteration, optimal policy (terminated, $ub(d) = n - 1$)	superlinear	superlinear	linear	linear
Q-learning, goal state reached	linear	sublinear	sublinear	sublinear
Q-learning, optimal policy (identified)	linear	linear	sublinear	sublinear
Q-learning, optimal policy (terminated, no $ub(d)$)	linear	linear	sublinear	sublinear
Q-learning, optimal policy (terminated, $ub(d) = d$)	linear	linear	sublinear	sublinear
Q-learning, optimal policy (terminated, $ub(d) = n - 1$)	superlinear	superlinear	linear	linear

Table 2: Shapes of complexity graphs

expect, in turn, to terminate earlier than a Q-learning algorithm with $ub(d) = ub_2$ for all $d \leq ub_1 < ub_2$. Similarly, we expect a value-iteration algorithm with $ub(d) = ub_1$ to terminate earlier than a value-iteration algorithm with $ub(d) = ub_2$ for all $d \leq ub_1 < ub_2$: We have proved that the bi-directional Q-learning algorithm (version 2) always terminates with an optimal policy no later than any bi-directional Q-learning algorithm (version 1), provided that ties are broken in the same way. Also, Q-learning and value-iteration terminate with an optimal policy the earlier, the smaller $ub(d)$, since then the earlier a U-value drops below $-ub(d)$, which terminates the algorithm.

- Furthermore, we expect the run-time of the efficient value-iteration algorithm to be smaller than the run-time of the efficient Q-learning algorithm which we expect to be smaller than the run-time of a random walk, given the same task to be solved and, for terminating with an optimal policy, the same $ub(d)$ used: Random walks do not remember the topology of the state space at all. Efficient value-iteration and efficient Q-learning both remember the topology partially, but value-iteration is more powerful, since it uses an action model.

- In addition to these relationships, the graphs show that random walks are inefficient in reset state spaces, but perform much better in one-dimensional and

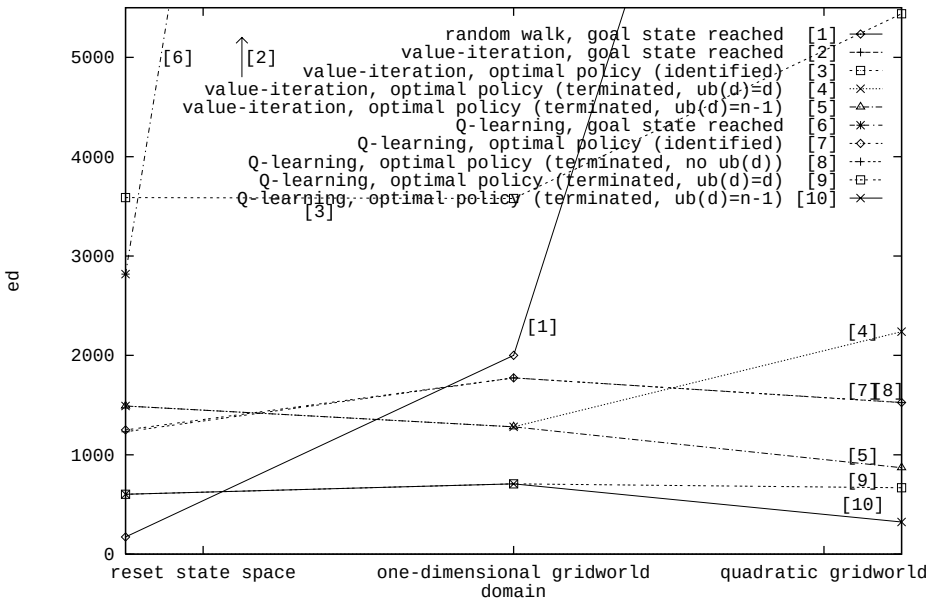

Figure 32: Complexity of domains that can be solved in 1000 steps

quadratic gridworlds, since the latter are Eulerian and therefore easier domains for random walks than the malicious reset state spaces, where the agent has to choose the correct action (out of two possible actions) $n-2$ times in a row in order to succeed. But even for gridworlds, the efficient Q-learning algorithms continue to perform better than random walks, since they remember information about the topology of the state space, whereas random walks do not.

8 Extensions

The results presented in this report can easily be extended to cases where the actions do not have the same reward, or prior knowledge of the topology of the state space is available.

- Prior knowledge (in form of suitable initial Q-values, i.e. consistent or admissible Q-values that are non-zero) makes the Q-learning algorithms better informed and can decrease their run-times, as can easily be seen from Lemma 2. For example, in the totally informed case, the Q-values are initialized as follows if action-penalty representation and no discounting are used:

$$Q(s,a) = \begin{cases} 0 & \text{if } s \in G \\ -1 - gd(succ(s,a)) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S \text{ and } a \in A(s)$$

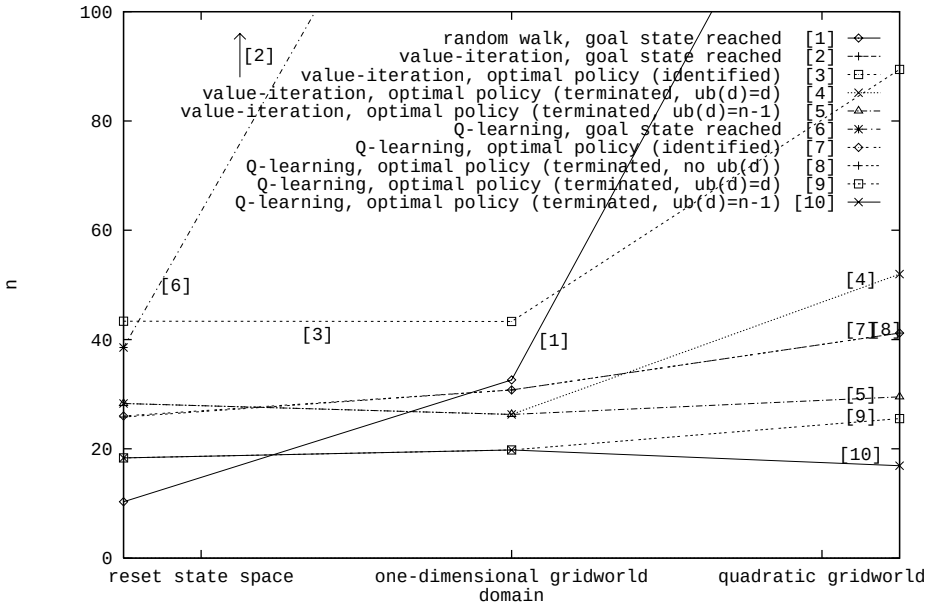

Figure 33: Sizes of domains that can be solved in 1000 steps

Lemma 2 predicts in this case that the agent needs only at most $-U(s) = gd(s)$ steps to reach a goal state from a given $s \in S$, the best number of steps possible.

In many cases, admissible heuristics (for A*-search) are known for the goal distances. As explained in Chapter 4.2.1, such heuristics allow one to initialize the Q-values with non-zero values and thus make the Q-learning algorithms better informed. Similarly, suitable initial U-values make the value-iteration algorithm better informed. Usually, the initial knowledge will be in between the two extremes of being totally uninformed and totally informed.

- Under the action-penalty representation, every action has an immediate cost of one. The results presented in this report can easily be adapted to the case where actions do not have the same immediate rewards, but arbitrary strictly negative immediate rewards. In this case, the complexity of reaching a goal state with admissible Q-learning (with an adequately adapted definition of admissibility) becomes $O(\frac{\overline{r}_{min}}{\overline{r}_{max}} ed)$ instead of $O(ed)$, where $\overline{r}_{min}$ is the smallest immediate reward (i.e. the one with the largest absolute value) and $\overline{r}_{max}$ is the largest immediate reward. If one defines the **weighted distance** $d_w(s, s')$ between $s \in S$ and $s' \in S$ to be the (unique) solution of the following set of equations

$$d_w(s,s') = \begin{cases} 0 & \text{if } s = s' \\ \min_{a \in A(s)}(-\overline{r}(s,a) + d_w(succ(s,a), s')) & \text{otherwise} \end{cases} \quad \text{for all } s, s' \in S,$$

and the **weighted depth of the state space** to be $d_w := \max_{s,s' \in S} d_w(s, s')$, then one can prove the even tighter bound $O(\frac{1}{|\overline{r}_{max}|} ed_w)$. (The proofs are analo-

gous to the proofs of Lemma 1, Lemma 2, and Theorem 2.) Similarly, the complexity of reaching a goal state with admissible value-iteration becomes $O(\frac{\overline{r}_{min}}{\overline{r}_{max}}nd)$ or, alternatively, $O(\frac{1}{|\overline{r}_{max}|}nd_w)$ instead of $O(nd)$.

9 Further Problems

Reinforcement learning algorithms can not only be used in deterministic state spaces, but also in state spaces with probabilistic action outcomes. The assumption usually made is that the relative complexities of different algorithms in deterministic domains predict their performance in probabilistic domains. However, three problems arise in the latter case that must be dealt with:

- In probabilistic domains, reinforcement learning algorithms do not only have to learn the potential outcomes of actions, but the transition probabilities as well, either explicitly (as done by value-iteration) or implicitly (as done by Q-learning). For Q-learning, the problem arises that the learning rate α can no longer be set to one, and one has to execute an action a in a state s repeatedly in order for $Q(s,a)$ to converge even if the U-values of all potential outcomes of a in s are already correct. This also implies that it can take a long time for Q-learning to decrease large initial Q-values to their final values, since the Q-values are only adjusted in small increments. Similarly, in case the transition probabilities are represented explicitly, a number of samples larger than one is needed to estimate each probability reliably. [11] and [32] propose heuristics that address these issues.

- In probabilistic domains, admissible Q-values do not necessarily remain admissible. This is due to the fact that the transition probabilities are unknown and only estimates are available. These estimates can deviate from the correct values a lot, although their difference approaches zero with probability one when the sample size goes towards infinity (provided that a reasonable estimation method is used). As a consequence, it can happen that $U(s) < U^{opt}(s)$ for a state s even if the U-values of the potential outcomes of all actions in s remain admissible. For example, consider undiscounted, zero-initialized value-iteration with action-penalty representation and assume that the transition frequencies are used to estimate the transition probabilities (i.e. maximum-likelihood estimation is used). Further assume that there is a state s with $A(s) = \{a\}$, executing a in s results in state s_1 with probability 0.5 and in state s_2 with the complementary probability, and $U^{opt}(s_1) < U^{opt}(s_2)$. If $U(s_1) = U^{opt}(s_1)$, $U(s_2) = U^{opt}(s_2)$, and executing a in s has resulted n_1 times in the successor state s_1 and n_2 times in the successor state s_2 where $n_1 > n_2$, then

$$U(s) \;=\; -1 + \frac{n_1}{n_1+n_2}U(s_1) + \frac{n_2}{n_1+n_2}U(s_2)$$

$$
\begin{aligned}
&= -1 + \frac{n_1}{n_1+n_2} U^{opt}(s_1) + \frac{n_2}{n_1+n_2} U^{opt}(s_2) \\
&< -1 + \frac{1}{2} U^{opt}(s_1) + \frac{1}{2} U^{opt}(s_2) \\
&= U^{opt}(s)
\end{aligned}
$$

after updating s. If $U(s) < U^{opt}(s)$, then s looks worse than it is, which can make the agent avoid executing actions that have s as an outcome. [11] calls this phenomenon "sticking".

- In probabilistic domains, optimal policies can have cycles (i.e. the agent visits the same state more than once with non-zero probability). This is another reason why it can take Q-learning (or value-iteration) a long time to decrease large initial Q-values (or U-values). It also implies that the average number of steps that are required to reach a goal state when executing the *optimal* policy can already be exponential in n [33]. In this case, exploration is clearly exponential and one has to factor out the inherent complexity of the state space from the complexity of the learning algorithm.

To summarize, more research is required to transfer the results from deterministic to probabilistic state spaces. This is part of our current research activities. However, the results reported here have already proved useful for an analysis of various extensions of reinforcement learning techniques. For example, [18] utilizes them to analyze the complexity of hierarchical Q-learning.

10 Conclusion

Many real-world domains have the characteristic of the task presented here — the agent must reach one of a number of goal states by taking actions, but the initial topology of the state space is unknown. Prior results which indicated that reinforcement learning algorithms performed random walks until they reach a goal state for the first time and therefore were exponential in n, the size of the state space, seemed to limit their usefulness for such tasks.

This report has shown, however, that such algorithms are tractable when using either an appropriate task representation (the action-penalty representation) or suitable initial Q-values. Both changes produce a dense reward structure, which facilitates learning. In particular, we showed that Q-learning needs at most $O(ed)$ steps to find a path from the start state to a goal state, or at most $O(n^3)$ steps if the state space has no duplicate actions. These bounds are tight. Moreover, *every* uninformed on-line search algorithm has the same big-O worst-case complexity. We showed, however, that learning (and subsequently using) a map of the state space can decrease the big-O worst-case complexity for *some* (but not all) domains: additional planning between action executions can reduce the number of action executions by more than a constant

factor if one is willing to tolerate an increase in deliberation time between action executions. But the complexity results also show that even if one knew the topology of the state space in advance and performed Q-learning anyway, one would only increase the number of action executions by a factor of $O(en)$ at most.

We have shown that both initial knowledge of the topology of the domain (in form of initial Q-values that are admissible, but non-zero) and domain properties such as having no identity actions, no duplicate actions, a constant individual upper action bound, a linear total upper action bound, polynomial width, being 1-step invertible or Eulerian can decrease the complexity of the Q-learning algorithm (even if the agent does not know whether a domain has these properties). Many reinforcement learning domains, for example gridworlds, share some or all of these properties. Therefore, exploration in these domains actually has very low complexity. For instance, the worst-case complexity of reaching a goal state with Q-learning in quadratic gridworlds is only $O(n^{3/2})$.

In general, the largest big-O average-case complexity of a random walk is much larger than the big-O worst-case complexity of Q-learning, and every uninformed on-line search algorithm has at least the same big-O worst-case complexity than Q-learning. For Eulerian state spaces, however, we have shown that the largest big-O average-case complexity of a random walk equals the big-O worst-case complexity of Q-learning. We continue to expect Q-learning to outperform a random walk (although the improvement can no longer be exponential) since random walks have no memory of past experiences. However, there exist uninformed on-line search algorithms that have a smaller big-O worst-case complexity than Q-learning. They demonstrate that actively utilizing properties of the state space can decrease the complexity. These results are *very* different from the general case. This shows that general results about the complexity of reinforcement learning algorithms might not be specific enough. It can be more interesting to identify specific properties of the reinforcement learning domain of interest and investigate how they influence the complexity.

We have introduced the novel *bi-directional Q-learning algorithm* for finding shortest paths from all states to a goal state and have shown, somewhat surprisingly, that its complexity is $O(ed)$ as well. This provides an efficient algorithm to learn optimal policies.

All results that we have derived in this report for Q-learning can easily be transferred to value-iteration, which uses an action model and can therefore be expected to be more efficient than Q-learning. Undiscounted, admissible value-iteration with action-penalty representation is equivalent to Korf's LRTA* algorithm with a search horizon of one. Our results generalize Korf's results to state spaces that contain identity actions and also show the effects of various domain properties and initial U-values that are non-zero on the complexity. We have shown that the behavior of Q-learning in any domain is identical to the behavior of value-iteration in an adequately modified domain. Our bi-directional value-iteration algorithm generalizes the LRTA* algorithm in that it finds shortest paths from *all* states to a goal state, does *not* need to have reset actions available, and terminates.

Tight bounds on the largest *average* number of steps required for reaching a goal state using a zero-initialized algorithm with goal-reward representation (the same results apply to finding optimal policies)

State Space	Q-Learning	Value-Iteration
general case	exponential	exponential
no duplicate actions	exponential	exponential
linear total upper action bound	exponential	exponential

Tight bounds on the number of steps required in the worst case for reaching a goal state using a zero-initialized algorithm with action-penalty representation or a one-initialized algorithm with goal-reward representation (the same results apply to finding optimal policies)

State Space	Q-Learning	Value-Iteration
general case	$O(en)$	$O(n^2)$
no duplicate actions	$O(n^3)$	$O(n^2)$
linear total upper action bound	$O(n^2)$	$O(n^2)$

Figure 34: Complexities of Reinforcement Learning

The important results (expressed in terms of n and e) are summarized in Figure 34. They demonstrate that undirected exploration methods can be tractable. This result is supported by our empirical studies in three different reinforcement learning domains, that also show that ed is a good measure for the complexity of a state space.

While some reinforcement learning tasks cannot be reformulated as shortest path problems (for example, non-goal-oriented tasks where the agent has to learn how to behave in the world), the theorems still provide guidance: the run-times can be improved by making the reward structure dense, for instance, by subtracting some constant $c \in \mathcal{R}$ from all immediate rewards. This does not change the relative preferences among plans, since

$$\begin{aligned}
\widehat{U}^{opt}(s) &< \widehat{U}^{opt}(s') \\
\sum_{t=0}^{\infty}(\gamma^t \widehat{\overline{r}}_t) &< \sum_{t=0}^{\infty}(\gamma^t \widehat{\overline{r}}'_t) \\
\sum_{t=0}^{\infty}(\gamma^t(\overline{r}_t - c)) &< \sum_{t=0}^{\infty}(\gamma^t(\overline{r}'_t - c)) \\
\sum_{t=0}^{\infty}(\gamma^t \overline{r}_t) - \frac{c}{1-\gamma} &< \sum_{t=0}^{\infty}(\gamma^t \overline{r}'_t) - \frac{c}{1-\gamma} \\
U^{opt}(s) - \frac{c}{1-\gamma} &< U^{opt}(s') - \frac{c}{1-\gamma} \\
U^{opt}(s) &< U^{opt}(s')
\end{aligned}$$

Alternatively, one can use sufficiently large initial Q-values for Q-learning (or U-values for value-iteration). In both cases it can be necessary to use discounting.

In summary, reinforcement learning algorithms are useful for enabling agents to explore unknown state spaces and learn information relevant to performing tasks. The results in this report add to that research by showing that reinforcement learning is tractable, and therefore can scale up to handle real-world problems.

11 Acknowledgments

The Reinforcement Learning Study Group at Carnegie Mellon University provided a stimulating research environment. Discussions with Sebastian Thrun initiated this research and provided valuable insight. Avrim Blum, Lonnie Chrisman, Long-Ji Lin, Michael Littman, Andrew Moore, Joseph O'Sullivan, Martha Pollack, and Sebastian Thrun provided helpful comments on the ideas presented in this report. Lonnie Chrisman, Michael Littman, and Joseph O'Sullivan read earlier drafts and provided helpful suggestions. In addition, Lonnie Chrisman provided detailed comments on the proofs in this report, which improved their presentation. Although we checked the report carefully, it will almost certainly still have inaccuracies. If you find one, please let us know.

References

[1] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundation of Computer Science*, pages 218–223, San Juan, Puerto Rico, 10 1979.

[2] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report 91–57, Department of Computer Science, University of Massachusetts at Amherst, 1991.

[3] Andrew G. Barto, R.S. Sutton, and C.J. Watkins. Learning and sequential decision making. Technical Report 89–95, Department of Computer Science, University of Massachusetts at Amherst, 1989.

[4] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[5] Gregory D. Benson and Armand Prieditis. Learning continuous-space navigation heuristics in real-time. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1992.

[6] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the FOCS*, 1990.

[7] William Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley and Sons, New York, London, Sydney, second edition, 1966.

[8] Ronald A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge (Massachusetts), third edition, 1964.

[9] Toru Ishida. Moving target search. In *Proceedings of the AAAI*, pages 525–532, 1992.

[10] Toru Ishida and Richard E. Korf. Moving target search. In *Proceedings of the IJCAI*, pages 204–210, 1991.

[11] Leslie P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Computer Science Department, Stanford University, 1990.

[12] Kevin Knight. Are many reactive agents better than a few deliberative ones? In *Proceedings of the IJCAI*, 1993.

[13] Sven Koenig. Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's thesis, Computer Science Department, University of California at Berkeley, 1991. (Available as Technical Report UCB/CSD 92/685).

[14] Sven Koenig. The complexity of real-time search. Technical Report CMU–CS–92–145, School of Computer Science, Carnegie Mellon University, 1992.

[15] Richard E. Korf. Real-time heuristic search: First results. In *Proceedings of the AAAI*, pages 133–138, 1987.

[16] Richard E. Korf. Real-time heuristic search: New results. In *Proceedings of the AAAI*, pages 139–144, 1988.

[17] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 3 1990.

[18] Long-Ji Lin. *Reinforcement Learning for Robots using Neural Networks.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1993. (Available as Technical Report CMU-CS-93-103).

[19] Andrew W. Moore and Christopher G. Atkeson. Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In *Proceedings of the NIPS*, 1992.

[20] Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, New York, 1971.

[21] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 8 1987.

[22] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Menlo Park, California, 1985.

[23] Joseph C. Pemberton and Richard E. Korf. Incremental path planning on graphs with cycles. In *Proceedings of the First Annual AI Planning Systems Conference*, pages 179–188, 1992.

[24] Jing Peng and Ronald J. Williams. Efficient learning and planning within the Dyna framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1992.

[25] Stuart Russell and Eric Wefald. *Do the Right Thing – Studies in Limited Rationality.* The MIT Press, Cambridge, Massachusetts, 1991.

[26] M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the IJCAI*, pages 1039–1046, 1987.

[27] Herbert A. Simon and Joseph B. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.

[28] Satinder P. Singh. Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the AAAI*, pages 202–207, 1992.

[29] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[30] Richard S. Sutton. First results with DYNA. In *Proceedings of the AAAI Spring Symposium*, 1990.

[31] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, 1990.

[32] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University, 1992.

[33] Sebastian B. Thrun. The role of exploration in learning control with neural networks. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky, 1992.

[34] Christopher J. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge University, 1989.

[35] Steven D. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings of the AAAI*, pages 607–613, 1991.

[36] Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, Computer Science Department, University of Rochester, 1991.

[37] Steven D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action*. PhD thesis, Department of Computer Science, University of Rochester, 1992.

[38] Richard Yee. Abstraction in control learning. Technical Report 92–16, Department of Computer Science, University of Massachusetts at Amherst, 1992.

Appendix

The appendix contains the proofs of the lemmas and theorems stated in the main text that have not been proved there. We use the terminology and symbols introduced in the main text.

A Upper Bounds

A.1 Reaching a Goal State with Q-Learning

In the following, we consider an undiscounted, admissible Q-learning algorithm with action-penalty representation. The proofs can then be transferred to the other algorithms (for example a discounted, admissible Q-learning algorithm with action-penalty representation or a high-initialized, discounted Q-learning algorithm with goal-reward representation) as outlined in the main text.

The time superscripts used in this chapter refer to the values of the variables immediately before the action execution step of the Q-learning algorithm, i.e. line 4, of step t.

Theorem 12 *Zero-initialized Q-values are consistent.*

Proof: $Q(s,a) = 0$ for all $s \in G$ and $a \in A(s)$. Since $U(s) = \max_{a \in A(s)} Q(s,a) = \max_{a \in A(s)} 0 = 0$ for all $s \in S$, it holds for all $s \in S \setminus G$ and $a \in A(s)$ that $-1 + U(succ(s,a)) = -1 + 0 \leq 0 = Q(s,a) \leq 0$.

Theorem 13 *(Minus one)-initialized Q-values are consistent.*

Proof: $Q(s,a) = 0$ for all $s \in G$ and $a \in A(s)$. Since $U(s) = \max_{a \in A(s)} Q(s,a) \leq \max_{a \in A(s)} 0 = 0$ for all $s \in S$, it holds for all $s \in S \setminus G$ and $a \in A(s)$ that $-1 + U(succ(s,a)) \leq -1 + 0 = Q(s,a) \leq 0$.

Theorem 14 *If Q-values are consistent, then* $-\min_{s' \in PG} d(s,s') \leq U(s) \leq 0$ *for all* $s \in S$, *where* $PG := \{s \in S : U(s) = 0\}$.

Proof by induction on $d'(s) := \min_{s' \in PG} d(s, s')$:

- If $d'(s) = 0$, then $s \in PG$ and therefore $U(s) = 0$. Thus, $-d'(s) = U(s) = 0$.
- If $d'(s) \neq 0$, then $s \notin PG \supseteq G$ and therefore $s \in S \setminus G$. Since $G \neq \emptyset$, it holds that $PG \neq \emptyset$. Let $a := \text{argmin}_{a' \in A(s)} d'(succ(s, a'))$. Assume that the theorem holds for all $s'' \in S$ with $d'(s'') < d'(s)$. Then, it holds for $succ(s, a)$ as well, since $d'(succ(s, a)) = d'(s) - 1 < d'(s)$. Thus, $-d'(s) = -1 - d'(succ(s, a)) \overset{\text{assumption}}{\leq} -1 + U(succ(s, a)) \leq Q(s, a) \leq \max_{a' \in A(s)} Q(s, a') = U(s) \leq \max_{s' \in S, a' \in A(s)} Q(s', a') \leq 0$.

Theorem 15 *If Q-values are consistent, then* $-1 - \min_{s' \in PG} d(succ(s, a), s') \leq Q(s, a) \leq 0$ *for all* $s \in S \setminus G$ *and* $a \in A(s)$, *where* $PG := \{s \in S : U(s) = 0\}$.

Proof: $-1 - \min_{s' \in PG} d(succ(s, a), s') \overset{\text{Theorem 14}}{\leq} -1 + U(succ(s, a)) \leq Q(s, a) \leq 0$ for all $s \in S \setminus G$ and $a \in A(s)$.

Theorem 16 *If Q-values are consistent, then* $-gd(s) \leq U(s) \leq 0$ *for all* $s \in S$.

Proof: $-gd(s) = -\min_{s' \in G} d(s, s') \leq -\min_{s' \in PG} d(s, s') \overset{\text{Theorem 14}}{\leq} U(s) \overset{\text{Theorem 14}}{\leq} 0$ for all $s \in S$.

Theorem 17 *If Q-values are consistent, then* $-d \leq \min_{s \in S} U(s) - \max_{s \in S} U(s)$.

Proof of $-d \leq -d'(s) \leq U(s) - \max_{s' \in S} U(s')$ for all $s \in S$ by induction on $d'(s) := \min_{s' \in G'} d(s, s')$, where $G' := \{s \in S : U(s) = \max_{s' \in S} U(s')\} \neq \emptyset$. Then, $-d \leq \min_{s \in S} U(s) - \max_{s \in S} U(s)$ and the theorem is proved.

- If $d'(s) = 0$, then $s \in G'$ and $-d'(s) = 0 = \max_{s' \in S} U(s') - \max_{s' \in S} U(s') = U(s) - \max_{s' \in S} U(s')$.
- If $d'(s) \neq 0$, then $U(s) < \max_{s' \in S} U(s') \overset{\text{Theorem 14}}{\leq} \max_{s' \in S} 0 = 0$ and therefore $s \notin G$. Assume that the theorem holds for all $s' \in S$ with $d'(s') < d'(s)$. There exists an $a \in A(s)$ such that $d'(succ(s, a)) = d'(s) - 1 < d'(s)$. Then, $-d'(s) = -1 - d'(succ(s, a)) \overset{\text{assumption}}{\leq} -1 + U(succ(s, a)) - \max_{s' \in S} U(s') \leq Q(s, a) - \max_{s' \in S} U(s') \leq \max_{a' \in A(s)} Q(s, a') - \max_{s' \in S} U(s') = U(s) - \max_{s' \in S} U(s')$.

Theorem 18 *Consistent Q-values are admissible.*

Proof: Assume consistent Q-values. $Q(s,a) = 0$ for all $s \in G$ and $a \in A(s)$. It holds for all $s \in S \setminus G$ and $a \in A(s)$ that $-1 - gd(succ(s,a)) \overset{\text{Theorem 16}}{\leq} -1 + U(succ(s,a)) \leq Q(s,a) \leq 0$.

Theorem 19 *$U^{opt}(s) = -gd(s)$ for all $s \in S$.*

Proof: According to Equations 2 and the fact that we are using action-penalty representation

$$U^{opt}(s) = \begin{cases} 0 & \text{if } s \in G \\ \max\limits_{a \in A(s)} (-1 + U^{opt}(succ(s,a))) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S$$

$$-U^{opt}(s) = \begin{cases} 0 & \text{if } s \in G \\ 1 + \min\limits_{a \in A(s)} -U^{opt}(succ(s,a)) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S$$

(We utilize that $U^{opt}(s) = 0$ for $s \in G$: Once in a goal state, the agent can execute only one action, that has an immediate reward of 0 and does not leave the goal state. Thus, the total reward of the continued execution of this action is 0. This fact does not follow from Equations 2 if $\gamma = 1$.)

Comparing this to the definition of $gd(s)$

$$\begin{aligned} gd(s) &= \min_{s' \in G} d(s,s') \\ &= \min_{s' \in G} \begin{cases} 0 & \text{if } s = s' \\ 1 + \min\limits_{a \in A(s)} d(succ(s,a),s') & \text{otherwise} \end{cases} \\ &= \begin{cases} 0 & \text{if } s \in G \\ 1 + \min\limits_{a \in A(s)} gd(succ(s,a)) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S \end{aligned}$$

shows that $-U^{opt}(s) = gd(s)$ for all $s \in S$, since the solution of the set of equations is unique.

Theorem 20 *If Q-values are admissible, then $-gd(s) \leq U(s) \leq 0$ for all $s \in S$.*

Proof: If $s \in G$, then $-gd(s) = 0 = \max_{a \in A(s)} 0 = \max_{a \in A(s)} Q(s,a) = U(s)$. It holds for all $s \in S \setminus G$ that $-gd(s) = -(1 + \min_{a \in A(s)} gd(succ(s,a))) = \max_{a \in A(s)}(-1 - gd(succ(s,a))) \leq \max_{a \in A(s)} Q(s,a) = U(s) \leq \max_{s' \in S, a' \in A(s)} Q(s',a') \leq 0$.

Consider the following algorithm: Given arbitrary Q-values. Pick an arbitrary state $s \in S \setminus G$ and determine $a := \text{argmax}_{a' \in A(s)} Q(s, a')$. (Ties can be broken arbitrarily.) Set $Q(s, a) := -1 + U(succ(s, a))$ and leave the other Q-values unchanged. Refer to the old Q-values as $Q^0(s, a)$ and to the new ones as $Q^1(s, a)$, i.e.

$$Q^1(s', a') = \begin{cases} -1 + U^0(succ(s', a')) & \text{if } s' = s \text{ and } a' = a \\ Q^0(s', a') & \text{otherwise} \end{cases} \qquad \text{for all } s' \in S \text{ and } a' \in A(s')$$

Theorem 21 *If the Q^0-values are consistent, then*

1. *$Q^1(s', a') \leq Q^0(s', a')$ for all $s' \in S$ and $a' \in A(s')$,*
2. *$U^1(s') \leq U^0(s')$ for all $s' \in S$, and*
3. *the Q^1-values are consistent.*

Proof:

1. $Q^1(s', a') = -1 + U^0(succ(s', a')) \leq Q^0(s', a')$ for $s' = s$ and $a' = a$, and $Q^1(s', a') = Q^0(s', a')$ otherwise. Thus, $Q^1(s', a') \leq Q^0(s', a')$ for all $s' \in S$ and $a' \in A(s')$.
2. According to the first part of this theorem, it holds that $Q^1(s', a') \leq Q^0(s', a') \leq 0$ for all $s' \in S$ and $a' \in A(s')$. Then, $U^1(s') = \max_{a' \in A(s')} Q^1(s', a') \leq \max_{a' \in A(s')} Q^0(s', a') = U^0(s')$ for all $s' \in S$.
3. According to the second part of this theorem, it holds that $U^1(s') \leq U^0(s')$ for all $s' \in S$. Then, $-1 + U^1(succ(s', a')) \leq -1 + U^0(succ(s', a')) = Q^1(s', a') \leq 0$ for $s' = s$ and $a' = a$, $Q^1(s', a') = Q^0(s', a') = 0$ for all $s' \in G$ and $a' \in A(s')$, and $-1 + U^1(succ(s', a')) \leq -1 + U^0(succ(s', a')) \leq Q^0(s', a') = Q^1(s', a') \leq 0$ otherwise. Thus, the Q^1-values are consistent.

Theorem 22 *If the initial Q-values are consistent, then they remain consistent after every step of the agent, and the Q-values and U-values are monotonically decreasing.*

Proof by induction on the number of steps: The Q-values are consistent before the first step. Assume that they are consistent before an arbitrary step. According to Theorem 21, they are consistent after the step, and the Q-values and U-values are monotonically decreasing.

Consider the following algorithm: Given arbitrary Q-values. Pick an arbitrary state $s \in S \setminus G$ and determine $a := \text{argmax}_{a' \in A(s)} Q(s, a')$. (Ties can be broken arbitrarily.)

Set $Q(s,a) := \min(Q(s,a), -1+U(succ(s,a)))$ and leave the other Q-values unchanged. Refer to the old Q-values as $Q^0(s,a)$ and to the new ones as $Q^1(s,a)$, i.e.

$$Q^1(s',a') = \begin{cases} \min(Q^0(s',a'), -1 + U^0(succ(s',a'))) & \text{if } s' = s \text{ and } a' = a \\ Q^0(s',a') & \text{otherwise} \end{cases} \qquad \text{for all } s' \in S \text{ and } a' \in A$$

Theorem 23 *If the Q^0-values are admissible, then*

1. *$Q^1(s',a') \leq Q^0(s',a')$ for all $s' \in S$ and $a' \in A(s')$,*
2. *$U^1(s') \leq U^0(s')$ for all $s' \in S$, and*
3. *the Q^1-values are admissible.*

Proof:

1. $Q^1(s',a') = \min(Q^0(s',a'), -1 + U^0(succ(s',a'))) \leq Q^0(s',a')$ for $s' = s$ and $a' = a$, and $Q^1(s',a') = Q^0(s',a')$ otherwise. Thus, $Q^1(s',a') \leq Q^0(s',a')$ for all $s' \in S$ and $a' \in A(s')$.
2. According to the first part of this theorem, it holds that $Q^1(s',a') \leq Q^0(s',a') \leq 0$ for all $s' \in S$ and $a' \in A(s')$. Then, $U^1(s') = \max_{a' \in A(s')} Q^1(s',a') \leq \max_{a' \in A(s')} Q^0(s',a') = U^0(s')$ for all $s' \in S$.
3. $-1 - gd(succ(s',a')) \leq Q^0(s',a')$ and $-1 - gd(succ(s',a')) \overset{\text{Theorem 20}}{\leq} -1 + U^0(succ(s',a'))$ for $s' = s$ and $a' = a$, therefore $-1 - gd(succ(s',a')) \leq \min(Q^0(s',a'), -1 + U^0(succ(s',a'))) = Q^1(s',a') \leq 0$ for $s' = s$ and $a' = a$. $Q^1(s',a') = Q^0(s',a') = 0$ for all $s' \in G$ and $a' \in A(s')$, and $-1 - gd(succ(s',a')) \leq Q^0(s',a') = Q^1(s',a') \leq 0$ otherwise. Thus, the Q^1-values are admissible.

Theorem 24 *If the initial Q-values are admissible, then they remain admissible after every step of the agent, and the Q-values and U-values are monotonically decreasing.*

Proof by induction on the number of steps: The Q-values are admissible before the first step. Assume that they are admissible before an arbitrary step. According to Theorem 23, they are admissible after the step, and the Q-values and U-values are monotonically decreasing.

Theorem 25 *For all steps $t \in \mathcal{N}_0$ (until termination) of an undiscounted, admissible Q-learning algorithm it holds that $U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) - t \geq \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) + U^0(s^0) - loop^t$ and $loop^t \leq \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a)$, where $loop^t := |\{t' \in \{0, \ldots, t-1\} : s^{t'} = s^{t'+1}\}|$ (the number of identity actions executed before t).*

Proof by induction on t: The theorem trivially holds for $t = 0$. Assume that it holds for an arbitrary t. Note that $Q^t(s^t, a^t) = U^t(s^t)$, due to the specific action-selection step used. We distinguish two cases:

- The action executed at t is an identity action:

  Then, $s^{t+1} = s^t$ and $loop^{t+1} = 1 + loop^t$. Depending on the value update step used, it holds that either $Q^{t+1}(s^t, a^t) = -1 + U^t(s^t) = -1 + Q^t(s^t, a^t)$ or $Q^{t+1}(s^t, a^t) = \min(Q^t(s^t, a^t), -1 + U^t(s^t)) = \min(Q^t(s^t, a^t), -1 + Q^t(s^t, a^t)) = -1 + Q^t(s^t, a^t)$. Thus, in both cases $Q^{t+1}(s^t, a^t) = -1 + Q^t(s^t, a^t)$. $U^{t+1}(s^t) = \max_{a \in A(s^t)} Q^{t+1}(s^t, a) \geq Q^{t+1}(s^t, a^t) = -1 + Q^t(s^t, a^t) = -1 + U^t(s^t)$. All other values do not change from t to $t + 1$.

  $U^{t+1}(s^{t+1}) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - (t + 1) \geq (-1 + U^t(s^t)) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - (t + 1) = (U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - t) - 2 \overset{\text{assumption}}{\geq} (\sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a) + U^0(s^0) - loop^t) - 2 = (-1 + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a)) + U^0(s^0) - (1 + loop^t) = \sum_{s \in S} \sum_{a \in A(s)} Q^{t+1}(s, a) + U^0(s^0) - loop^{t+1}$.

  $loop^{t+1} = 1 + loop^t \overset{\text{assumption}}{\leq} 1 + (\sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a)) = \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - (-1 + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a)) = \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - \sum_{s \in S} \sum_{a \in A(s)} Q^{t+1}(s, a)$.

  In other words, the theorem also holds for $t + 1$.

- The action executed at t is not an identity action:

  Then, $s^{t+1} \neq s^t$, $loop^{t+1} = loop^t$, and (independent of the value update step used) $Q^{t+1}(s^t, a^t) \leq -1 + U^t(s^{t+1}) = -1 + U^{t+1}(s^{t+1})$. All other values, except for $U(s^t)$, do not change from t to $t + 1$.

  $U^{t+1}(s^{t+1}) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - (t + 1) = (U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - t) + U^{t+1}(s^{t+1}) - U^t(s^t) - 1 \overset{\text{assumption}}{\geq} (\sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a) + U^0(s^0) - loop^t) + U^{t+1}(s^{t+1}) - U^t(s^t) - 1 = (-1 + U^{t+1}(s^{t+1})) - U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a) + U^0(s^0) - loop^t \geq (Q^{t+1}(s^t, a^t) - Q^t(s^t, a^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a)) + U^0(s^0) - loop^t = (\sum_{s \in S} \sum_{a \in A(s)} Q^{t+1}(s, a)) + U^0(s^0) - loop^{t+1}$.

  $loop^{t+1} = loop^t \overset{\text{assumption}}{\leq} \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a) \leq \sum_{s \in S} \sum_{a \in A(s)} Q^0(s, a) - \sum_{s \in S} \sum_{a \in A(s)} Q^{t+1}(s, a)$, since $Q^{t+1}(s, a) \leq Q^t(s, a)$ for all $s \in S$ and $a \in A(s)$ according to Theorems 22 and 24.

  In other words, the theorem also holds for $t + 1$.

The following theorem is a simplified version of the previous one.

Theorem 26 *For all steps $t \in \mathcal{N}_0$ (until termination) of an undiscounted, zero-initialized, and admissible Q-learning algorithm it holds that $U^t(s^t) - t = \sum_{s \in S} \sum_{a \in A(s)} Q^t(s, a)$ if the state space has no identity actions.*

Proof by induction on t: The theorem trivially holds for $t = 0$. Assume that it holds for an arbitrary t. $s^{t+1} \neq s^t$ and (independent of the value update step used) $Q^{t+1}(s^t, a^t) = -1 + U^t(s^{t+1}) = -1 + U^{t+1}(s^{t+1})$, since the Q^t-values are consistent according to Theorems 12 and 22 and therefore $Q^t(s^t, a^t) \geq -1 + U^t(succ(s^t, a^t))$. All other values, except for $U(s^t)$, do not change from t to $t+1$. Note that $Q^t(s^t, a^t) = U^t(s^t)$, due to the specific action-selection step used.

$U^{t+1}(s^{t+1}) - (t+1) = (U^t(s^t) - t) + U^{t+1}(s^{t+1}) - U^t(s^t) - 1 \stackrel{\text{assumption}}{=} \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) + U^{t+1}(s^{t+1}) - U^t(s^t) - 1 = (-1 + U^{t+1}(s^{t+1})) - U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) = Q^{t+1}(s^t, a^t) - Q^t(s^t, a^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) = \sum_{s \in S} \sum_{a \in A(s)} Q^{t+1}(s,a)$.

In other words, the theorem also holds for $t+1$.

Theorem 27 *For all steps $t \in \mathcal{N}_0$ (until termination) of an undiscounted, admissible Q-learning algorithm it holds that $t \leq U^t(s^t) - U^0(s^0) + 2 \sum_{s \in S} \sum_{a \in A(s)} (Q^0(s,a) - Q^t(s,a))$.*

Proof: $t \stackrel{\text{Theorem 25}}{\leq} U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) - U^0(s^0) + loop^t \stackrel{\text{Theorem 25}}{\leq} U^t(s^t) + \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a) - U^0(s^0) + (\sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) - \sum_{s \in S} \sum_{a \in A(s)} Q^t(s,a)) = U^t(s^t) + 2 \sum_{s \in S} \sum_{a \in A(s)} Q^0(s,a) + 2 \sum_{s \in S} \sum_{a \in A(s)} -Q^t(s,a) - U^0(s^0) = U^t(s^t) - U^0(s^0) + 2 \sum_{s \in S} \sum_{a \in A(s)} (Q^0(s,a) - Q^t(s,a))$.

Theorem 28 *An undiscounted, admissible Q-learning algorithm reaches a goal state and terminates after at most $2 \sum_{s \in S \setminus G} \sum_{a \in A(s)} (Q^0(s,a) + gd(succ(s,a)) + 1) - U^0(s^0)$ steps.*

Proof: $t \stackrel{\text{Theorem 27}}{\leq} U^t(s^t) - U^0(s^0) + 2 \sum_{s \in S} \sum_{a \in A(s)} (Q^0(s,a) - Q^t(s,a)) \leq -U^0(s^0) + 2 \sum_{s \in S \setminus G} \sum_{a \in A(s)} (Q^0(s,a) + gd(succ(s,a)) + 1)$, since the Q-values are admissible according to Theorems 22 and 18 or Theorem 24, and therefore $Q^0(s,a) = Q^t(s,a) = 0$ for all $s \in G$ and $a \in A(s)$, $-1 - gd(succ(s,a)) \leq Q^t(s,a)$ for all $s \in S \setminus G$ and $a \in A(s)$, and $U^t(s) \stackrel{\text{Theorem 20}}{\leq} 0$ for all $s \in S$.

Theorem 29 *An admissible Q-learning algorithm reaches a goal state and terminates after at most $O(ed)$ steps.*

Proof: The algorithm reaches a goal state and terminates after at most $O(2 \sum_{s \in S \setminus G} \sum_{a \in A(s)} (Q^0(s,a) + gd(succ(s,a)) + 1) - U^0(s^0)) \leq O(2 \sum_{s \in S \setminus G} \sum_{a \in A(s)} (d +$

$1) + d) \leq O(2e(d+1)+d) = O(ed)$ steps according to Theorem 28, since the Q-values are admissible according to Theorems 22 and 18 or Theorem 24, and therefore $Q^0(s,a) \leq 0$ for all $s \in S$ and $a \in A(s)$, and $-d \leq -gd(s) \overset{\text{Theorem 20}}{\leq} U^0(s)$ for all $s \in S$.

A.2 Finding Optimal Policies with Bi-Directional Q-Learning (Version 1)

In the following, we consider the bi-directional Q-learning algorithm (version 1) as stated in Figure 20. The proofs can then be transferred to a discounted, admissible bi-directional Q-learning algorithm with action-penalty representation or a high-initialized, discounted bi-directional Q-learning algorithm with goal-reward representation as outlined in Chapter 4 for the (basic) Q-learning algorithm.

The time superscripts used in this chapter refer to the values of the variables immediately before the action execution steps of the bi-directional Q-learning algorithm, i.e. lines 5 and 9, of step t.

We define a "forward phase" of the bi-directional Q-learning algorithm (version 1) to be a largest interval of steps $[t^a, t^b] \subseteq \mathcal{N}_0$ (with $t^a < t^b$) such that line 5 was executed at all steps $t \in [t^a, t^b - 1]$. Analogously, we define a "backward phase" to be a largest interval of steps $[t^a, t^b] \subseteq \mathcal{N}_0$ (with $t^a < t^b$) such that line 9 was executed at all steps $t \in [t^a, t^b - 1]$.

These definitions imply that the agent is in a state s^{t^a} with $done^{t^a}(s^{t^a}) = \mathit{false}$ at the beginning of a forward phase $[t^a, t^b]$. Then, it exclusively executes forward steps, until it finally reaches a state s^{t^b} with $done^{t^b}(s^{t^b}) = \mathit{true}$. The opposite holds for a backward phase.

The bi-directional Q-learning algorithm starts with a forward phase if $s_{start} \notin G$, then alternates between backward and forward phases, and ends with a backward phase.

Theorem 30 *The Q_f-values are consistent after every step of the agent and are monotonically decreasing.*

Proof by induction on t:

- Initially, $Q_f(s,a) = 0$ for all $s \in S$ and $a \in A(s)$. Thus, the Q_f-values are consistent according to Theorem 12.
- Assume that the Q_f-values are consistent before an arbitrary step. The only line that can change a Q_f-value is line 6. If line 6 is executed, then the Q_f-values

remain consistent after the step and are monotonically decreasing according to Theorem 21. If line 6 is not executed, then the Q_f-values remain unchanged and are consistent according to the assumption.

Theorem 31 *Every forward phase terminates.*

Proof by contradiction: Assume not. The Q_f-values before the first step of any forward phase are consistent according to Theorem 30. The repeated execution of a forward step implements an admissible Q-learning algorithm. According to Theorem 29, the agent reaches a state $s \in G$ eventually, and then sets $done(s,a) := true$ for all $a \in A(s)$ in line 2. Afterwards, $done(s) = true$ and the next step executed is a backward step. This terminates the forward phase, which is a contradiction.

Theorem 32 *For all $s \in S$ and $a \in A(s)$, it holds that: once $done(s,a) = true$, it remains true after every step of the agent and $Q_f(s,a)$ remains unchanged.*

Proof by contradiction: Assume not. Then $done(s,a) = true$ before some step, but after the step either the value of $done(s,a)$ or $Q_f(s,a)$ has changed. The only line that can set $done(s,a) := false$ or change the value of $Q_f(s,a)$ is line 6 when the current state equals s. It is only executed if $done(s) = false$ when line 3 is executed. This implies that $done(s,a') = false$ for all $a' \in A(s)$ with $Q_f(s,a') = \max_{a'' \in A(s)} Q_f(s,a'')$. But then $done(s,a) = false$ for the action a selected in line 4, which is a contradiction.

Theorem 33 *For all $s \in S$, it holds that: once $done(s) = true$, it remains true after every step of the agent and $U_f(s)$ remains unchanged.*

Proof by contradiction: Assume not. Then $done(s) = true$ before some step, but after the step either the value of $done(s)$ or $U_f(s)$ has changed. This is only possible if a value of $done(s,a)$ or $Q_f(s,a)$ for an $a \in A(s)$ has changed. The only line that can change one of these values is line 6 when the current state equals s, but it cannot be executed, since $done(s) = true$ when line 3 is executed. This is a contradiction.

Theorem 34 *Every forward phase $[t^a, t^b]$ increases the cardinality of the set $D := \{(s,a) : done(s,a) = true \vee done(s) = true, s \in S, a \in A(s)\}$ by at least one.*

Proof: $done^{t^b}(s^{t^b}) = true$ per definition of a forward phase. We distinguish two cases:

- $done^{t^b-1}(s^{t^b}) = \mathit{false}$:

  $A(s^{t^b}) \neq \emptyset$, since the state space is strongly connected. $done^{t^b-1}(s^{t^b}) = \mathit{false}$ implies that there exists an $a \in A(s^{t^b})$ with $done^{t^b-1}(s^{t^b}, a) = \mathit{false}$. Thus, $(s^{t^b}, a) \notin D^{t^b-1}$. However, $(s^{t^b}, a) \in D^{t^b}$, since $done^{t^b}(s^{t^b}) = \mathit{true}$.

- $done^{t^b-1}(s^{t^b}) = \mathit{true}$:

  If a forward step is executed at step $t^b - 1 \geq t^a$, then $done^{t^b-1}(s^{t^b-1}) = \mathit{false}$ (otherwise a backward step would be executed, which were a contradiction). Thus, $done^{t^b-1}(s^{t^b-1}, a^{t^b-1}) = \mathit{false}$ (otherwise $done^{t^b-1}(s^{t^b-1}) = \exists_{a \in A(s^{t_b-1})}(Q_f^{t^b-1}(s^{t^b-1}, a) = \max_{a' \in A(s^{t^b-1})} Q_f^{t^b-1}(s^{t^b-1}, a') \wedge done(s^{t^b-1}, a)) = \mathit{true}$, since $Q_f^{t^b-1}(s^{t^b-1}, a^{t^b-1}) = \max_{a' \in A(s^{t^b-1})} Q_f^{t^b-1}(s^{t^b-1}, a')$ and $done(s^{t^b-1}, a^{t^b-1}) = \mathit{true}$, which were a contradiction). Thus, $(s^{t^b-1}, a^{t^b-1}) \notin D^{t^b-1}$. However, $(s^{t^b-1}, a^{t^b-1}) \in D^{t^b}$, since $done^{t^b}(s^{t^b-1}, a^{t^b-1}) = done^{t^b-1}(succ(s^{t^b-1}, a^{t^b-1})) = done^{t^b-1}(s^{t^b}) = \mathit{true}$.

Theorem 35 *For all $s \in S$, it holds that: if $done(s) = \mathit{true}$, then $U_f(s) = -gd(s) = U_f^{opt}(s)$.*

Proof by induction on t: Initially, $done(s) = \mathit{false}$ for all $s \in S$ and the theorem holds trivially. We distinguish two cases:

- $gd(s) = 0$, i.e. $s \in G$:

  Initially, $U_f(s) = \max_{a \in A(s)} Q_f(s, a) = \max_{a \in A(s)} 0 = 0 = -gd(s) \overset{\text{Theorem 19}}{=} U_f^{opt}(s)$. The only line that can change a $Q_f(s, a)$ value for $a \in A(s)$ is line 6 when the current state equals s, but it cannot be executed, since $done(s, a)$ is set to true for all $a \in A(s)$ in line 2 and thus $done(s) = \mathit{true}$ when line 3 is executed. Thus, the $Q_f(s, a)$ values cannot be changed for all $a \in A(s)$, and $U_f(s) = \max_{a \in A(s)} Q_f(s, a) = \max_{a \in A(s)} 0 = 0 = -gd(s) = U_f^{opt}(s)$ continues to hold after every step of the agent (no matter what the value of $done(s)$ is).

- $gd(s) \neq 0$, i.e. $s \in S \setminus G$:

  If $done(s)$ never becomes true, the theorem holds trivially for $s \in S$. Otherwise, let $t := \text{argmin}_{t' \in \mathcal{N}_0}(done^{t'}(s) = \mathit{true})$ and assume that the theorem holds for all steps smaller than t. Because $done^t(s) = \mathit{true}$, there exists an $a \in A(s)$ with $Q_f^t(s, a) = \max_{a' \in A(s)} Q_f^t(s, a')$ and $done^t(s, a) = \mathit{true}$. Since initially $done(s, a) = \mathit{false}$, there exists a step $t' \in \mathcal{N}_0$ with $t' < t$ and $done^{t'}(s, a) = \mathit{false}$, but $done^{t'+1}(s, a) = \mathit{true}$. Line 6 was executed at step t', since this is the only way to set $done(s, a) := \mathit{true}$ for a state $s \notin G$. Then, $done^{t'+1}(s, a) = done^{t'}(succ(s, a)) = \mathit{true}$ and $Q_f^{t'+1}(s, a) = -1 + U_f^{t'}(succ(s, a))$. $-1 - gd(succ(s, a)) \leq \max_{a' \in A(s)}(-1 - gd(succ(s, a'))) =$

$-(1+\min_{a' \in A(s)} gd(succ(s,a'))) = -gd(s) \overset{\text{Theorem 16}}{\leq} U_f^t(s) = \max_{a' \in A(s)} Q_f^t(s,a') = Q_f^t(s,a) \overset{\text{Theorem 30}}{\leq} Q_f^{t'+1}(s,a) = -1 + U_f^{t'}(succ(s,a)) \overset{\text{assumption}}{=} -1 - gd(succ(s,a))$, since the Q_f-values are consistent at step t according to Theorem 30. Thus, equality holds and $U_f^t(s) = -gd(s) \overset{\text{Theorem 19}}{=} U_f^{opt}(s)$. Since $done^t(s) = true$, $U_f(s)$ remains unchanged after every step of the agent and $U_f(s) = -gd(s) = U_f^{opt}(s)$ continues to hold according to Theorem 33.

Theorem 36 *For all $s \in S \setminus G$ and $a \in A(s))$, it holds that: if $done(s,a) = true$, then $Q_f(s,a) = -1 - gd(succ(s,a))$.*

Proof: Initially, $done(s,a) = false$. If $done(s,a)$ never becomes true, the theorem holds trivially for $s \in S \setminus G$. Otherwise, let $t := \text{argmin}_{t \in \mathcal{N}_0}(done^t(s,a) = true)$. Since $s \notin G$, the only line that can set $done(s,a) := true$ is line 6 when the current state equals s. Then, $done^t(s,a) = done^{t-1}(succ(s,a))$ and $Q_f^t(s,a) = -1 + U_f^{t-1}(succ(s,a))$. Since $done^{t-1}(succ(s,a)) = done^t(s,a) = true$, Theorem 35 asserts that $U_f^{t-1}(succ(s,a)) = -gd(succ(s,a))$. It follows that $Q_f^t(s,a) = -1 - gd(succ(s,a))$. Since $done^t(s,a) = true$, $Q_f(s,a)$ remains unchanged after every step of the agent and $Q_f(s,a) = -1 - gd(succ(s,a))$ continues to hold according to Theorem 32.

Theorem 37 *There is a maximum of e forward phases and $e+1$ backward phases.*

Proof: According to Theorem 34, every forward phase increases the size of the set $D := \{(s,a) : done(s,a) = true \vee done(s) = true, s \in S, a \in A(s)\}$ by at least one. Since $|D| = e$, there can be at most e forward phases. Since forward and backward phases alternate, there are at most $e+1$ backward phases.

Theorem 38 *All forward phases together execute at most $O(e \times ub(d))$ steps.*

Proof: There are at most e forward phases according to Theorem 37. Let there be the forward phases $[t_i^a, t_i^b]$ for $i \in \{0, 1, \ldots, e'\}$ with $e' < e$. $Q_f^{t_i^b}(s,a) = Q_f^{t_{i+1}^a}(s,a)$ and $U_f^{t_i^b}(s) = U_f^{t_{i+1}^a}(s)$ for all $i \in \{0, 1, \ldots, e'-1\}$, $s \in S$, and $a \in A(s)$, since only forward steps can change Q_f-values. According to Theorem 31, every forward phase terminates. Theorem 27 asserts that the forward phase $[t_i^a, t_i^b]$ needs at most $U_f^{t_i^b}(s^{t_i^b}) - U_f^{t_i^a}(s^{t_i^a}) + 2\sum_{s \in S}\sum_{a \in A(s)}(Q_f^{t_i^a}(s,a) - Q_f^{t_i^b}(s,a))$ steps to terminate. The following relationships hold, since the Q_f-values are consistent after every step of the agent according to Theorem 30 and $G \neq \emptyset$: $-d \leq -gd(s) \overset{\text{Theorem 16}}{\leq} U_f(s) \overset{\text{Theorem 16}}{\leq} 0$ for all

$s \in S$, $-1-d \leq -1-gd(succ(s,a)) \leq Q_f(s,a) \leq 0$ for all $s \in S \setminus G$ and $a \in A(s)$, and $-1-d \leq 0 = Q_f(s,a)$ for $s \in G$ and $a \in A(s)$. Thus, all forward phases together terminate after at most $\sum_{i=0}^{e'} \left(U_f^{t_i^b}(s^{t_i^b}) - U_f^{t_i^a}(s^{t_i^a}) + 2 \sum_{s \in S} \sum_{a \in A(s)} (Q_f^{t_i^a}(s,a) - Q_f^{t_i^b}(s,a)) \right) = \sum_{i=0}^{e'} (U_f^{t_i^b}(s^{t_i^b}) - U_f^{t_i^a}(s^{t_i^a})) + 2 \sum_{s \in S} \sum_{a \in A(s)} (Q_f^{t_0^a}(s,a) - Q_f^{t_{e'}^b}(s,a)) \leq \sum_{i=0}^{e'} (0 + d)) + 2 \sum_{s \in S} \sum_{a \in A(s)} (0 + (d+1)) \leq ed + 2e(d+1) \leq 3e(d+1) \leq 3e(ub(d)+1)$ steps in total, which are $O(e \times ub(d))$ steps.

Theorem 39 *The Q_b-values are consistent after every step of the agent and are monotonically decreasing.*

Proof by induction on t: Define $G_b := \{s \in S \setminus G : done(s) = false\}$. Thus, G_b changes during execution.

- Initially, $Q_b(s,a) = 0$ for all $s \in S$ and $a \in A(s)$. Thus, the Q_b-values are consistent according to Theorem 12.
- Assume that the Q_b-values are consistent for the set of goal states G_b^t before an arbitrary step $t+1$. Note that if Q-values are consistent for a set of goal states G, then they are also consistent for any subset $G' \subseteq G$. According to Theorem 33, it holds that $G_b^{t+1} \subseteq G_b^t$. Thus, the Q_b^t-values are consistent for the set of goal states G_b^{t+1} as well.

  The only line that can change a Q_b-value is line 10. If line 10 is not executed, then the Q_b-values remain unchanged and therefore remain consistent for G_b^{t+1}. If line 10 is executed, then $done^t(s^t) = true$ (otherwise line 3 would have transferred control to a forward step) and therefore $s^t \notin G_b^t \supseteq G_b^{t+1}$. Then, the Q_b-values remain consistent for G_b^{t+1} after the step and are monotonically decreasing according to Theorem 21.

Theorem 40 *$U_f(s) = -gd(s) = U_f^{opt}(s)$, $done(s) = true$, and $U_b(s) < 0$ for all $s \in S$ after termination if the bi-directional Q-learning algorithm (version 1) terminates.*

Proof: Let t be the time superscript for the final values of the variables. Then, $U_b^t(s^{t-1}) < -ub(d) \leq -d$, since the condition in line 11 is true at step t. According to Theorem 39, the Q_b^t-values are consistent. Theorem 17 states that $-d \leq \min_{s' \in S} U_b^t(s') - \max_{s' \in S} U_b^t(s') \leq U_b^t(s^{t-1}) - \max_{s' \in S} U_b^t(s') < -d - \max_{s' \in S} U_b^t(s')$. Thus, $0 = -d + d < -d - \max_{s' \in S} U_b^t(s') + d = -\max_{s' \in S} U_b^t(s')$. Consider an arbitrary $s \in S$. Since $\max_{s' \in S} U_b^t(s') < 0$, it must hold that $U_b^t(s) < 0$, but initially $U_b(s) = 0$. Thus, all values $Q_b(s,a)$ for $a \in A(s)$ have changed. Since only line 10 can change these values, it is executed at least once with the current state equal to s. At this

point in time, $done(s) = true$ (otherwise line 10 could not have been executed) and, according to Theorem 33, $done^t(s) = true$. Then, $U_f^t(s) = -gd(s) = U_f^{opt}(s)$ according to Theorem 35.

Theorem 41 *It is an optimal policy to select action* $argmax_{a \in A(s)} U_f(succ(s,a))$ *or, equivalently,* $argmax_{a \in A(s), done(s,a)=true} Q_f(s,a)$ *in state* $s \in S \setminus G$ *after termination if the bi-directional Q-learning algorithm (version 1) terminates ("correctness").*

Proof: Let t be the time superscript for the final values of the variables. We prove both parts of the theorem separately:

- Consider an arbitrary $s \in S \setminus G$ and any $a \in A(s)$ with $U_f^t(succ(s,a)) = \max_{a' \in A(s)} U_f^t(succ(s,a'))$. Since $U_f^t(s') = -gd(s')$ for all $s' \in S$ according to Theorem 40, it holds that $gd(succ(s,a)) = \min_{a' \in A(s)} gd(succ(s,a'))$. Thus, it is optimal to execute a in s.
- Now, consider an arbitrary $s \in S \setminus G$ and any $a \in A(s)$ with $done^t(s,a) = true$ and $Q_f^t(s,a) = \max_{a'' \in A(s), done^t(s,a'')=true} Q_f^t(s,a'')$. Since $done^t(s) = true$, there exists an $a' \in A(s)$ with $done^t(s,a') = true$ and $Q_f^t(s,a') = \max_{a'' \in A(s)} Q_f^t(s,a'')$. Thus, $Q_f^t(s,a) = \max_{a'' \in A(s), done^t(s,a'')=true} Q_f^t(s,a'') = Q_f^t(s,a') = \max_{a'' \in A(s)} Q_f^t(s,a'') \geq Q_f^t(s,a''')$ for all $a''' \in A(s)$. Since $done^t(s,a) = true$, $Q_f^t(s,a) = -1 - gd(succ(s,a))$ according to Theorem 36. Furthermore, the Q_f^t-values are consistent according to Theorem 30 and therefore $-1 - gd(succ(s,a''')) \leq Q_f^t(s,a''')$. Thus, $gd(succ(s,a)) = -1 - Q_f^t(s,a) \leq -1 - Q_f^t(s,a''') \leq gd(succ(s,a'''))$ for all $a''' \in A(s)$. Since $gd(succ(s,a)) = \min_{a'' \in A(s)} gd(succ(s,a''))$, it is optimal to execute a in s.

Theorem 42 *Every backward phase terminates.*

Proof by contradiction: Assume not. Then there must a state s that is visited infinitely often. We call the sequence of states between two occurrences of state s a cycle. Theorem 39 asserts that the Q_b-values before the first step of the backward phase are consistent. The repeated execution of a backward step implements an admissible Q-learning algorithm. Thus, the Q_b-values are monotonically decreasing. Furthermore, for every cycle that the agent completes, the largest Q_b-value of the actions executed in the cycle decreases by at least one. Eventually, the Q_b-values of all actions executed in the cycle drop below every bound, so do the U_b-values. In particular, they drop below $-ub(d)$, the condition in line 11 is satisfied, and the backward phase terminates, which is a contradiction. (For more details, see similar arguments in the context of RTA*-type search by [17] or [25].)

Theorem 43 *All backward phases together execute at most $O(e \times ub(d))$ steps.*

Proof: There are at most $e+1$ backward phases according to Theorem 37. Let there be the backward phases $[t_i^a, t_i^b]$ for $i \in \{0, 1, \ldots, e'\}$ with $e' < e+1$, where we set $t_{e'}^b$ to the step before termination (i.e. one smaller than it should be). This simplifies the notation in the following.

$Q_b^{t_i^b}(s,a) = Q_b^{t_{i+1}^a}(s,a)$ and $U_b^{t_i^b}(s) = U_b^{t_{i+1}^a}(s)$ for all $i \in \{0, 1, \ldots, e'-1\}$, $s \in S$, and $a \in A(s)$, since only backward steps can change Q_b-values. According to Theorem 42, every backward phase terminates. Theorem 27 asserts that the backward phase $[t_i^a, t_i^b]$ needs at most $U_b^{t_i^b}(s^{t_i^b}) - U_b^{t_i^a}(s^{t_i^a}) + 2\sum_{s\in S}\sum_{a\in A(s)}(Q_b^{t_i^a}(s,a) - Q_b^{t_i^b}(s,a))$ steps to terminate. Note that $-ub(d) \leq U_b(s) \leq 0$ for all $s \in S$ and all steps but the final step (otherwise the algorithm would have terminated earlier), and therefore $-1 - ub(d) \leq -1 - U_b^{t_{e'}^b}(succ(s,a)) \leq Q_b^{t_{e'}^b}(s,a)$ for all $s \in S \setminus G$ and $a \in A(s)$. Thus, all backward phases together terminate after at most $\sum_{i=0}^{e'}\left(U_b^{t_i^b}(s^{t_i^b}) - U_b^{t_i^a}(s^{t_i^a}) + 2\sum_{s\in S}\sum_{a\in A(s)}(Q_b^{t_i^a}(s,a) - Q_b^{t_i^b}(s,a))\right) = \sum_{i=0}^{e'}(U_b^{t_i^b}(s^{t_i^b}) - U_b^{t_i^a}(s^{t_i^a})) + 2\sum_{s\in S}\sum_{a\in A(s)}(Q_b^{t_0^a}(s,a) - Q_b^{t_{e'}^b}(s,a)) \leq \sum_{i=0}^{e'}(0 + ub(d)) + 2\sum_{s\in S}\sum_{a\in A(s)}(0 + (ub(d)+1)) \leq (e+1) \times ub(d) + 2e(ub(d)+1) \leq 3(e+1)(ub(d)+1)$ steps in total (plus the last step), which are $O(e \times ub(d))$ steps.

Theorem 44 *The bi-directional Q-learning algorithm (version 1) finds an optimal policy and terminates after at most $O(e \times ub(d))$ steps.*

Proof: According to Theorems 38 and 43, the forward and backward phases together execute at most $O(e \times ub(d))$ steps. According to Theorem 41, the bi-directional Q-learning algorithm terminates with an optimal policy.

A.3 Finding Optimal Policies with Bi-Directional Q-Learning (Version 2)

In the following, we consider the bi-directional Q-learning algorithm (version 2) as stated in Figure 21.

The time superscripts used in this chapter refer to the values of the variables immediately before the action execution steps of the bi-directional Q-learning algorithm, i.e. lines 7 and 12, of step t.

We define $\hat{S}$ to be the set of states that the agent *has* already explored, i.e. $\hat{S}^t := \{s \in S : \exists_{t' \leq t} s^{t'} = s\}$. Let $\hat{A}(s) \subseteq A(s)$ be the set of actions in s that the agent *has not* yet

explored, i.e. $\hat{A}^t(s) := \{a \in A(s) : \neg\exists_{t'<t}(s^{t'} = s \wedge a^{t'} = a)\}$. Note the asymmetries: First, $\hat{S}$ is the set of *explored* states, but $\hat{A}(s)$ is a set of *unexplored* actions. This asymmetry simplifies the notation. Second, $t' \leq t$ in the definition of $\hat{S}^t$, but $t' < t$ in the definition of $\hat{A}^t(s)$. This is so, because a step ends with the execution of an action (according to the definition of "step"). Thus, the current state of the agent before the action execution is already explored, since the agent is already in the state. However, the action selected for execution is not, since it has not been executed yet. Thus, we say that an action that has not been executed before step t, but is executed at step t, was unexplored (before and) at step t and is explored at step $t+1$.

Theorem 45 *For all $s \in S$, $a \in A(s)$, and steps $t \in \mathcal{N}_0$ (until termination), it holds that: $Q_f^t(s,a) = 0$ iff a has never been executed in s in a forward step before step t, i.e. for all $t' < t$ it has never been true that $s^{t'} = s$, $a^{t'} = a$, and $done^{t'}(s) = false$.*

Proof: Initially, $Q_f^0(s,a) = 0$. Only the execution of a in s in a forward step can change $Q_f(s,a)$. Thus, if a has never been executed in s in a forward step before step t, then $Q_f^t(s,a) = 0$. However, if a has been executed in s in a forward step at least once, say at step $t' < t$, then $Q_f^t(s,a) \overset{\text{Theorem 30}}{\leq} Q_f^{t'+1}(s,a) = -1 + U_f^{t'}(succ(s,a)) \overset{\text{Theorem 16}}{\leq} -1+0 = -1 < 0$.

Theorem 46 *For all $s \in S$, $a \in A(s)$, and steps $t \in \mathcal{N}_0$ (until termination), it holds that: $Q_b^t(s,a) = 0$ iff a has never been executed in s in a backward step before step t, i.e. for all $t' < t$ it has never been true that $s^{t'} = s$, $a^{t'} = a$, and $done^{t'}(s) = true$.*

Proof: Initially, $Q_b^0(s,a) = 0$. Only the execution of a in s in a backward step can change $Q_b(s,a)$. Thus, if a has never been executed in s in a backward step before step t, then $Q_b^t(s,a) = 0$. However, if a has been executed in s in a backward step at least once, say at step $t' < t$, then $Q_b^t(s,a) \overset{\text{Theorem 30}}{\leq} Q_b^{t'+1}(s,a) = -1 + U_b^{t'}(succ(s,a)) \overset{\text{Theorem 16}}{\leq} -1 + 0 = -1 < 0$.

Theorem 47 *For all $s \in S$ and steps $t \in \mathcal{N}_0$ (until termination), it holds that: $Q_f^{t+1}(s,a) = Q_b^{t+1}(s,a) = 0$ for all $a \in A(s)$ iff s is unexplored at step t, i.e. $s \notin \hat{S}^t$.*

Proof: If $Q_f^{t+1}(s,a) = Q_b^{t+1}(s,a) = 0$ for all $a \in A(s)$, then, according to Theorems 45 and 46, no $a \in A(s)$ has ever been executed in s before step $t+1$ (neither in a forward step nor in a backward step). Thus, $s \notin \hat{S}^t$, because otherwise the agent had been in state s at step t or before and would have had to execute an action in it, which is a contradiction. Likewise, if $s \notin \hat{S}^t$, then the agent has never been in

s at step t or before, and thus had no chance to execute an action in it. Therefore, $Q_f^{t+1}(s,a) = Q_b^{t+1}(s,a) = 0$ for all $a \in A(s)$ according to Theorems 45 and 46.

Theorem 48 *For all steps $t \in \mathcal{N}_0$ (until termination), $memory^t = |\{(s,a) : s \in \hat{S}^t \cap G, a \in \hat{A}^t(s)\}| + |\{s \in \hat{S}^t \setminus G : \neg done^t(s)\}|$.*

Proof by induction on t:

- Before the execution of the algorithm (i.e. at "step -1"), $memory = 0$ and $\hat{S} = \emptyset$. Thus, $memory = 0 = |\{(s,a) : s \in \hat{S} \cap G, a \in \hat{A}(s)\}| + |\{s \in \hat{S} \setminus G : \neg done(s)\}|$.
- Assume that the equation holds for step t.

  The only lines that can change the value of $memory$ are line 3, line 9, and line 14. Likewise, there are only three ways how $|\{(s,a) : s \in \hat{S} \cap G, a \in \hat{A}(s)\}| + |\{s \in \hat{S} \setminus G : \neg done(s)\}|$ can change between steps t and $t+1$: $\hat{S}$ can change, $done(s)$ for one or more $s \in \hat{S}^t \setminus G$ can change, or $\hat{A}(s)$ for one or more $s \in \hat{S}^t \cap G$ can change. We show that three equivalence relationships hold:

  - The body of the condition on line 3 is executed iff $\hat{S}$ has changed between steps t and $t+1$.

    If the body of the condition on line 3 is executed, then $Q_f^{t+1}(s^{t+1},a) = Q_b^{t+1}(s^{t+1},a) = 0$ for all $a \in A(s^{t+1})$, i.e. $s^{t+1} \notin \hat{S}^t$ according to Theorem 47. Since $s^{t+1} \in \hat{S}^{t+1}$, $\hat{S}$ has changed between steps t and $t+1$. After the execution of line 3, $memory$ is increased by $|A(s^{t+1})|$ if $s^{t+1} \in G$, otherwise by one.

    Assume now that $\hat{S}$ has changed between steps t and $t+1$, i.e. $s^{t+1} \notin \hat{S}^t$ and $\hat{S}^{t+1} = \hat{S}^t \cup \{s^{t+1}\}$. Thus, no action has yet been executed in s^{t+1}, i.e. $\hat{A}^{t+1}(s^{t+1}) = A(s^{t+1})$, and therefore $Q_f^{t+1}(s^{t+1},a) = Q_b^{t+1}(s^{t+1},a) = 0$ for all $a \in A(s^{t+1})$ according to Theorem 47. Thus, the body of the condition on line 3 is executed. If $s^{t+1} \in G$, then $|\{(s,a) : s \in \hat{S} \cap G, a \in \hat{A}(s)\}| + |\{s \in \hat{S} \setminus G : \neg done(s)\}|$ increases by $|A(s^{t+1})|$, since $|\hat{A}^{t+1}(s^{t+1})| = |A(s^{t+1})|$. If $s^{t+1} \notin G$, then $|\{(s,a) : s \in \hat{S} \cap G, a \in \hat{A}(s)\}| + |\{s \in \hat{S} \setminus G : \neg done(s)\}|$ increases by one, since $done^{t+1}(s^{t+1}) = false$: initially $done(s^{t+1}) = false$, only line 8 can change $done(s^{t+1})$ (because $s^{t+1} \notin G$), and line 8 cannot have been executed (because $s^{t+1} \notin \hat{S}^t$).
  - The body of the condition on line 9 is executed iff $done(s)$ for one or more $s \in \hat{S}^t \setminus G$ has changed between steps t and $t+1$.

    If the body of the condition on line 9 is executed, then $done(s^t)$ has changed. $s^t \in \hat{S}^t \setminus G$, because otherwise line 2 would set $done(s^t) := true$ and line 5 would give control to a backward step, which is a contradiction. Thus, $done(s)$ for one or more $s \in \hat{S}^t \setminus G$ has changed between steps t and $t+1$. After the execution of line 9, $memory$ is decreased by one.

Assume now that $done(s)$ for one or more $s \in \hat{S}^t \setminus G$ has changed between steps t and $t+1$. Only line 8 can change $done(s)$ for $s \notin G$. Thus, $done(s^t)$ has changed, and the body of the condition on line 9 is executed. At most one $done(s)$ for $s \in \hat{S}^t \setminus G$ can change between steps t and $t+1$, because line 8 is the only line that can change such a value. According to Theorem 33, $done(s)$ can only change from *false* to *true* for all $s \in S$. Thus, $|\{s \in \hat{S}^t \setminus G : \neg done(s)\}|$ is decreased by one, since $done(s^t)$ has changed from *false* to *true*, but all other $done(s)$ stayed the same.

- The body of the condition on line 14 is executed iff $\hat{A}(s)$ for one or more $s \in \hat{S}^t \cap G$ has changed between steps t and $t+1$.

  If the body of the condition on line 14 is executed, then $s^t \in \hat{S}^t \cap G$ and $Q_b(s^t, a^t)$ has changed from zero to non-zero. $a^t \notin \hat{A}^{t+1}(s^t)$, since a^t was executed in s^t in a backward step at step t. According to Theorem 30, the Q_f^t-values are admissible, which implies that $Q_f^t(s,a) = 0$ for all $s \in G$ and $a \in A(s)$. In particular, $Q_f^t(s^t, a^t) = 0$. Then, $Q_b^t(s^t, a^t) = 0$ implies $a^t \in \hat{A}^t(s^t)$ according to Theorems 45 and 46. Since $a^t \in \hat{A}^t(s^t)$, but $a^t \notin \hat{A}^{t+1}(s^t)$, $\hat{A}(s)$ for one or more $s \in \hat{S}^t \cap G$ has changed between steps t and $t+1$. After the execution of line 14, *memory* is decreased by one.

  Assume now that $\hat{A}(s)$ for one or more $s \in \hat{S}^t \cap G$ has changed between steps t and $t+1$. Since only line 12 can execute actions in a goal state s (line 2 sets $done(s) := true$ and thus line 5 always gives control to a backward step), it follows that $a^t \in \hat{A}^t(s^t)$ and $\hat{A}^{t+1}(s^t) = \hat{A}^t(s^t) \setminus \{a^t\}$, all other $\hat{A}(s)$ stay the same. Since $a^t \in \hat{A}^t(s^t)$, $Q_b^t(s^t, a^t) = 0$ according to Theorem 46. Since a^t was executed in s^t in a backward step at step t, $Q_b^{t+1}(s^t, a^t) \neq 0$ according to Theorem 46. Then, $Q_b(s^t, a^t)$ changed from zero to non-zero and the body of the condition on line 14 is executed. Also, $|\{(s,a) : s \in \hat{S}^t \cap G, a \in \hat{A}(s)\}|$ is decreased by one.

Since the equation stated in the theorem holds for step t according to the assumption and we have shown that between steps t and step $t+1$ the left-hand side of the equation changes as much as its right-hand side, it follows that the equation of the theorem holds for step $t+1$ as well.

Theorem 49 $0 \leq memory^t \leq e$ *for all steps* $t \in \mathcal{N}_0$ *(until termination).*

Proof: According to Theorem 48, it holds that $memory^t = |\{(s,a) : s \in \hat{S}^t \cap G, a \in \hat{A}^t(s)\}| + |\{s \in \hat{S}^t \setminus G : \neg done^t(s)\}|$. $0 \leq |\{(s,a) : s \in \hat{S}^t \cap G, a \in \hat{A}^t(s)\}| + |\{s \in \hat{S}^t \setminus G : \neg done^t(s)\}| \leq \sum_{s \in G} |A(s)| + |S \setminus G| \leq \sum_{s \in S} |A(s)| = e$, since the state space is strongly connected and therefore $|A(s)| \geq 1$ for all $s \in S$, i.e. $|S \setminus G| \leq \sum_{s \in S \setminus G} |A(s)|$.

Theorem 50 *$U_f(s) = -gd(s) = U_f^{opt}(s)$ and $done(s) = true$ for all $s \in S$ after termination if the bi-directional Q-learning algorithm (version 2) terminates. Furthermore, it is an optimal policy to select action $argmax_{a \in A(s)} U_f(succ(s,a))$ or, equivalently, $argmax_{a \in A(s), done(s,a)=true} Q_f(s,a)$ in state $s \in S \setminus G$ after termination ("correctness").*

Proof by contradiction: Let t be the time superscript for the final values of the variables. Assume that it does not hold that $done^t(s) = true$ for all $s \in S$. Then, there exists an $s \in S$ with $done^t(s) = false$. We distinguish two cases:

- $s \in \hat{S}^t$:

  $s \in \hat{S}^t$ and $done^t(s) = false$ implies according to Theorem 48 that $memory^t \geq 1$. However, $memory = 0$ upon termination, which is a contradiction.

- $s \notin \hat{S}^t$:

  s can be reached from every state in $\hat{S}^t \neq \emptyset$, since the state space is strongly connected. Thus, $\hat{A}^t(s') \neq \emptyset$ for at least one $s' \in \hat{S}^t$. Assume $a \in \hat{A}^t(s')$. If $s' \in G$, then $memory^t \geq 1$ according to Theorem 48, since $|\hat{A}^t(s')| > 0$. Assume $s' \notin G$. Then, $-gd(s') < 0 \stackrel{\text{Theorem 46}}{=} Q_b^t(s',a) \leq \max_{a' \in A(s')} Q_b^t(s',a') = U_b^t(s')$. This implies according to Theorem 35 that $done^t(s') = false$. Thus, $memory^t \geq 1$ according to Theorem 48, since $s' \in \hat{S}^t$, $s' \notin G$, and $done^t(s') = false$. In both cases, $memory^t \geq 1$. However, $memory = 0$ upon termination, which is a contradiction.

Thus, $done^t(s) = true$ and, according to Theorem 35, $U_f^t(s) = -gd(s) = U_f^{opt}(s)$ for all $s \in S$. The proof of Theorem 41 relies only on this fact. Thus, according to Theorem 41, it is an optimal policy to select action $\text{argmax}_{a \in A(s)} U_f^t(succ(s,a))$ or, equivalently, $\text{argmax}_{a \in A(s), done(s,a)=true} Q_f^t(s,a)$ in state $s \in S \setminus G$.

Theorem 51 *The bi-directional Q-learning algorithm (version 2) terminates at the earliest time t at which $U_b^t(s^{t-1}) < -d$ or before.*

Proof: Assume that $U_b(s) < -d$ after the value updating in a backward step. Note that this is exactly the situation in which the bi-directional Q-learning algorithm (version 1) terminates. The following two things follow. First, according to Theorem 40, $done(s') = true$ for all $s' \in S$. Second, also according to Theorem 40, $U_b(s') < 0$ for all $s' \in S$. This implies that $Q_b(s',a) \leq \max_{a' \in A(s')} Q_b(s',a') = U_b(s') < 0$ for all $s' \in S$ and $a \in A(s')$. Thus, all actions have been executed at least once according to Theorem 46. Put together, it follows that $done(s') = true$ and $\hat{A}(s') = \emptyset$ for all $s' \in S$. Once $done(s') = true$, it remains *true* according to Theorem 33. Similarly, $\hat{A}(s')$ stays $\emptyset$ once it is empty. Thus, next time line 4 is reached, $memory = 0$ according to Theorem 48, and the algorithm terminates.

Theorem 52 *The bi-directional Q-learning algorithm (version 2) finds an optimal policy and terminates after at most $O(ed)$ steps.*

Proof: Version 1 and version 2 of the bi-directional Q-learning algorithm differ only in their termination criteria. According to Theorem 44, the bi-directional Q-learning algorithm (version 1) terminates after at most $O(ed)$ steps if $ub(d) = d$. Let t be the time superscript for the values of the variables upon termination of the bi-directional Q-learning algorithm (version 1). Then, $U_b^t(s^{t-1}) < -ub(d) \leq -d$. Then, the bi-directional Q-learning algorithm (version 2) terminates no later than step t according to Theorem 51. Thus, it finds an optimal policy and terminates after at most $O(ed)$ steps. According to Theorem 50, it terminates with an optimal policy.

A.4 Value-Iteration

All theorems about the value-iteration algorithm or the bi-directional value-iteration algorithm can be proved analogous to their Q-learning counterparts. Therefore, we omit the proofs here. Some of the proofs can be found in [14]. It contains proofs for the theorems about reaching a goal state with an admissible value-iteration algorithm if the initial Q-values are *consistent*, and finding optimal policies with a version of the bi-directional value-iteration algorithm that is slightly different from the one presented in this report. With only minor changes, the proofs also apply to the more general case of reaching a goal state with an admissible value-iteration algorithm if the initial Q-values are *admissible*.

A.5 Reaching a Goal State with Random Walks in Eulerian State Spaces

Consider a random walk that starts in $s_{start} \in S$ and continues forever. Let P denote the steady state probabilities of the random walk. Formally,

$$P_s := \lim_{t \to \infty} \frac{E(\text{number of times the random walk visits } s \text{ in the first } t \text{ steps})}{t} \quad \text{for all } s \in S$$

i.e. P_s is the average probability in the long run that the random walk is in s. P_s is well-defined and independent of s_{start}, since the state space is strongly connected. It holds that

$$\sum_{s \in S} P_s = 1 \tag{5}$$

$$P_s = \sum_{s' \in S} \sum_{a \in A(s'): succ(s',a)=s} \frac{P_{s'}}{|A(s')|} \qquad \text{for all } s \in S \tag{6}$$

For all $s, s' \in S$, let $T(s, s')$ denote the expected number of steps that the random walk needs to enter s' for the first time if $s_{start} = s$.

Theorem 53 (Aleliunas et al. [1]) $\frac{P_s}{|A(s)|} = \frac{P_{s'}}{|A(s')|}$ *for all* $s, s' \in S$ *if the state space is Eulerian.*

Proof by contradiction: Assume not. Then, there exist $s, s' \in S$ and $a \in A(s')$ with $succ(s', a) = s$ such that both $\frac{P_s}{|A(s)|} = \max_{s'' \in S} \frac{P_{s''}}{|A(s'')|}$ and $\frac{P_{s'}}{|A(s')|} < \max_{s'' \in S} \frac{P_{s''}}{|A(s'')|}$.

Rewriting Equation 6 yields $\frac{P_s}{|A(s)|} = \frac{\sum_{s'' \in S} \sum_{a \in A(s'') : succ(s'', a) = s} \frac{P_{s''}}{|A(s'')|}}{|A(s)|}$. Since the state space is Eulerian, it holds that $|A(s)| = |\{(s'', a) : succ(s'', a) = s \wedge s'' \in S \wedge a \in A(s'')\}|$. Thus, $\frac{P_s}{|A(s)|}$ is the average over $\frac{P_{s''}}{|A(s'')|}$ of all predecessors s'' of s. If one averages over a couple of numbers all of which are smaller than or equal to a number i and one of the numbers is strictly smaller than i, then the average is strictly smaller than i, too. Since $\frac{P_{s'}}{|A(s')|} < \max_{s'' \in S} \frac{P_{s''}}{|A(s'')|}$ for the predecessor s' of s, it follows that $\frac{P_s}{|A(s)|} < \max_{s'' \in S} \frac{P_{s''}}{|A(s'')|}$, which is a contradiction.

Theorem 54 (Aleliunas et al. [1]) $P_s = \frac{|A(s)|}{e}$ *for all* $s \in S$ *if the state space is Eulerian.*

Proof: $1 \stackrel{\text{Equation 5}}{=} \sum_{s' \in S} P_{s'} = \sum_{s' \in S} (\frac{P_{s'}}{|A(s')|} |A(s')|) \stackrel{\text{Theorem 53}}{=} \sum_{s' \in S} (\frac{P_s}{|A(s)|} |A(s')|) = \frac{P_s}{|A(s)|} \sum_{s' \in S} |A(s')| = \frac{P_s}{|A(s)|} e$ for all $s \in S$. Thus, $P_s = \frac{|A(s)|}{e}$.

Theorem 55 (Aleliunas et al. [1]) $T(s, succ(s, a)) \leq e$ *for all* $s \in S$ *and* $a \in A(s)$ *if the state space is Eulerian.*

Proof: $P_s \stackrel{\text{Theorem 54}}{=} \frac{|A(s)|}{e}$. Thus, the average probability that the random walk is in s in the long run equals $\frac{|A(s)|}{e}$. The probability that it executes a when it is in s is $\frac{1}{|A(s)|}$. The joint probability that it is in s in the long run *and* executes a is $\frac{|A(s)|}{e} \frac{1}{|A(s)|} = \frac{1}{e}$. The expected number of steps between two occurrences of the agent being in s and executing a is the reciprocal of its probability in the long run, i.e. it is e. It follows that $T(s, succ(s, a)) \leq e$.

Theorem 56 $T(s, s') \leq ed$ *for all* $s, s' \in S$ *if the state space is Eulerian.*

Proof: There is a path from s to s' of length at most d. The expected number of steps that a random walk needs to traverse this path is the sum of the expected number of steps to go from every state on the path to its successor, each of which is at most e according to Theorem 55. Thus, $T(s, s') \leq ed$.

Theorem 57 *A zero-initialized Q-learning algorithm with goal-reward representation reaches a goal state and terminates after at most $O(ed)$ steps on average if the state space is Eulerian.*

Proof: A zero-initialized Q-learning algorithm with goal-reward representation performs a random walk until it reaches a goal state for the first time. The expected number of steps needed to reach a goal state and terminate is $x_{s_{start}} \leq \min_{s \in G} T(s_{start}, s)$. If the state space is Eulerian, then $\min_{s \in G} T(s_{start}, s) \overset{\text{Theorem 56}}{\leq} \min_{s \in G} ed = ed$. Thus, the average-case complexity is at most $O(x_{s_{start}}) \leq O(ed)$.

B Lower Bounds

We prove all lower bounds on the worst-case complexity of reaching a goal state by giving an example of a trace (i.e. state sequence) that achieves or tops the bound. The traces are specified as pseudo-code that, when executed, prints the state sequence. The `for-to`-loops have a default increment of one and the `for-downto`-loops have a default increment of minus one, unless specified otherwise with a `step`-clause. The statements that belong to the body of a loop are indicated by indentation. They are not executed if the range of the loop variable is empty. For example, the pseudo-code given in Chapter B.1 specifies the trace $1234 \ldots n$, that has length $n - 1$ (i.e. needs $n - 1$ steps to execute).

B.1 Reaching a Goal State with any Search Algorithm

Figure 18 shows a state space for which every search algorithm needs at least $n - 1$ steps to reach the goal state (for $n \geq 1$):

```
for i := 1 to n
  print i
```

B.2 Reaching a Goal State with Uninformed On-Line Search Algorithms

The following proofs of lower bounds that hold for *all* uninformed on-line search algorithms utilize that the agent cannot distinguish the goodness of unexplored actions in the current state. Similarly, the proofs of lower bounds that hold for *all* search algorithms that have to enter a state at least once before they know the successor states utilize that the agent cannot distinguish among unexplored successor states of the current state. In both cases, we supply a tie breaking rule that, when followed by the agent, achieves or tops the lower bound. For all domains in this report, this rule is to prefer actions that lead to states with smaller numbers. Every uninformed on-line search algorithm can traverse a supersequence of the traces given by us.

Figure 11 shows a domain for which every uninformed on-line search algorithm can need at least $(e-n+1)(n-1)$ steps to reach the goal state (for $n \geq 2$ and $e \geq n$):

```
for i := 1 to e-n+1
  for j := 1 to n-1
    print j
print n
```

Figure 14 shows a domain for which every uninformed on-line search algorithm can need at least $(e-n)(n-2)+1$ steps to reach the goal state (for $n \geq 3$ and $e \geq n+1$):

```
print 1
for i := 1 to e-n
  for j := 2 to n-1
    print j
print n
```

An off-line search algorithm that knows the topology of the state space can reach the goal state in one step:

```
print 1
print n
```

Figure 15 shows a domain for which every uninformed on-line search algorithm can need at least $1/6n^3 - 1/6n$ steps to reach the goal state (for $n \geq 1$):

```
for i := 1 to n-1
  print i
  for j := 1 to i-1
    for k := j to i
      print k
print n
```

This fact and Theorem 2 together imply that the complexity of an admissible, zero-initialized Q-learning algorithm in the domain shown in Figure 15 is tight at $O(n^3)$.

Figure 19 shows a domain for which every uninformed on-line search algorithm can need at least $e + n - 4$ steps to reach the goal state (for $n > 2$ and $e \geq 2n - 2$):

```
for i := 1 to (e-2n+4)/2
  print 2
  print 1
for i := 2 to n-2
  print i
  print i+1
  print i
print n-1
print n
```

Figure 23 shows a domain for which every algorithm that has to enter a state at least once before it knows the successor states can need at least $1/2n^2 - 1/2n$ steps to reach the goal state (for $n \geq 1$):

```
for i := 1 to n-1
  for j := i downto 1
    print j
print n
```

An off-line search algorithm that knows the topology of the state space can reach the goal state in one step:

```
print 1
print n
```

Figure 25 shows a domain for which every algorithm that has to enter a state at least once before it knows the successor states can need at least $1/4n^2 - 1$ steps to reach the goal state (for even $n > 1$):

```
for i:= 1 to n-3 step 2
  for j := 1 to i step 2
    print j
  for j := i+1 downto 2 step 2
    print j
for i := 1 to n-1 step 2
  print i
```

B.3 Reaching a Goal State with Q-Learning and Value-Iteration

Because an admissible, zero-initialized Q-learning algorithm is an uninformed on-line search algorithm, the lower bounds proved above for uninformed on-line search algorithms also hold for Q-learning. Similarly, since an admissible, zero-initialized value-iteration algorithm is an algorithm that has to enter a state at least once before it knows the successor states, the lower bounds proved above for these algorithms also hold for value-iteration. In the following, we provide traces for the domains used in the main text that are not covered by the above proofs or for which we can prove a larger lower bound. (See chapter B.4 for an analysis of the domain shown in Figure 12.)

Figure 17 shows a domain for which an admissible, zero-initialized Q-learning algorithm can need at least $1/16n^3 + 3/8n^2 - 3/16n - 1/4$ steps to reach the goal state (for $n \geq 1$ with $n \bmod 4 = 1$):

```
for i := n-1 downto (3n+1)/4
  for j := (n+1)/2 downto 1
    print j
  for j := 2 to (n+1)/2
    print j
    for k := 1 to j-2
      print k
      print j
  for j := (n+3)/2 to i-1
    print j
  for j := i downto (n+3)/2
    print j
for j := (n+1)/2 downto 1
  print j
for j := 2 to (n+1)/2
  print j
  for k := 1 to j-2
```

```
    print k
    print j
for i := (n+3)/2 to n
  print i
```

Figure 19 shows a domain for which an admissible, zero-initialized Q-learning algorithm can need at least $1/2en - 1/4n^2 - 1/2n + 2$ steps to reach the goal state (for even $n > 2$ and $e \geq 2n - 2$):

```
for i := n-1 downto (n+2)/2
  for j := 1 to (e-2n+4)/2
    print 2
    print 1
  for j:= 2 to i-1
    print j
  for j:= i downto 3
    print j
for i := 1 to (e-2n+4)/2
  print 2
  print 1
for i := 2 to n
  print i
```

Figure 22 shows a domain for which an admissible, zero-initialized value-iteration algorithm can need at least $n^2 - n$ steps to reach the goal state (for $n \geq 1$):

```
for i := 1 to n-1
  for j := 1 to i
    print i
  for j := i downto 1
    print j
print n
```

An off-line search algorithm that knows the topology of the state space can reach the goal state in one step:

```
print 1
print n
```

Figure 24 shows a domain for which an admissible, zero-initialized value-iteration algorithm can need at least $3/16n^2 - 3/4$ steps to reach the goal state (for $n \geq 1$ with $n \bmod 4 = 2$):

```
for i := n-3 downto n/2 step 2
  for j := 1 to i step 2
    print j
  for j := i+1 downto 2 step 2
    print j
for i := 1 to n-1 step 2
  print i
```

B.4 Reaching a Goal State – A More Complicated Case

Consider an undiscounted, admissible, and zero-initialized Q-learning algorithm that operates in a reset state space of size $n \geq 1$. As shown in Figure 8, the actions a_1 decrease the goal distance, whereas the actions a_2 lead the agent back to state 1.

Theorem 58 *$U(s) \leq \frac{s-n}{2}$ for all $s \in S$ after termination of an undiscounted, admissible, and zero-initialized Q-learning algorithm in a reset state space of size $n \geq 1$ if ties are broken in favor of actions that lead to states with smaller numbers.*

Proof by induction on n. Let t' be the step when the algorithm has terminated, i.e. $U^{t'}(s)$ are the final U-values.

- If $n = 1$, then the algorithm terminates immediately, since $s_{start} = 1 \in G$. All Q-values remain zero, and $U^{t'}(s) = \max_{a \in A(s)} Q^{t'}(s,a) = \max_{a \in A(s)} 0 = 0 = \frac{s-1}{2}$ for all $s \in S = \{1\}$.

- Assume that the theorem holds for an arbitrary $n \geq 1$ and consider now a reset state space of size $n+1$.

  When the agent enters state $n+1$, the algorithm terminates before changing any Q-value in that state. The values $Q(n+1,a)$ remain zero for all $a \in A(n+1)$, and $U^{t'}(n+1) = \max_{a \in A(n+1)} Q^{t'}(n+1,a) = \max_{a \in A(n+1)} 0 = 0 = \frac{(n+1)-(n+1)}{2}$.

  State $n+1$ is the only goal state. In order to reach it from $s_{start} = 1$, the agent has to visit state n at least once. Let $t < t'$ be the earliest step at which $s^t = n$. Since the states $s \leq n$ form (together with their actions that do not leave this set of states) a reset state space of size n, it holds that $U^t(s) \leq \frac{s-n}{2}$ for all $s \in \{1, 2, \ldots, n\}$ according to the assumption. Since $Q^t(n, a_1) = Q^t(n, a_2) = 0$,

$succ(n, a_2) = 1 < n+1 = succ(n, a_1)$, and ties are broken in favor of actions that lead to states with smaller numbers, the agent chooses $a^t = a_2$ for execution at step t and is in state $s^{t+1} = 1$ afterwards. In order to reach the goal state $n+1$ from there, the agent has to execute action a_1 in every state $s \in \{1, 2, \ldots, n\}$ at least once. Let $t' > t(s) > t$ be a step at which $s^{t(s)} = s \in \{1, 2, \ldots, n\}$ and $a^{t(s)} = a_1$.

The following argument holds for all $s \in \{2, 3 \ldots, n-1\}$: $Q^{t(s)}(s, a_1) > Q^{t(s)}(s, a_2)$, since a_1 is chosen over a_2 at step $t(s)$. (If $Q^{t(s)}(s, a_1) \leq Q^{t(s)}(s, a_2)$, then a_2 would be chosen over a_1, since $succ(s, a_2) = 1 < s+1 = succ(s, a_1)$ and ties are broken in favor of actions that lead to states with smaller numbers.) Since the Q-values are integers, $Q^{t'}(s, a_2) \stackrel{\text{Theorem 22}}{\leq} Q^{t(s)}(s, a_2) \leq Q^{t(s)}(s, a_1) - 1 \stackrel{\text{Theorem 22}}{\leq} Q^t(s, a_1) - 1 \leq \max(Q^t(s, a_1), Q^t(s, a_2)) - 1 = U^t(s) - 1 \leq \frac{s-n}{2} - 1 \leq \frac{s-n-2}{2}$. $Q^{t'}(s, a_1) \stackrel{\text{Theorem 22}}{\leq} Q^{t(s)+1}(s, a_1) = -1 + U^{t(s)}(succ(s, a_1)) \stackrel{\text{Theorem 22}}{\leq} -1 + U^t(s+1) \leq -1 + \frac{(s+1)-n}{2} = \frac{s-n-1}{2}$. Then, $U^{t'}(s) = \max(Q^{t'}(s, a_1), Q^{t'}(s, a_2)) \leq \max(\frac{s-n-1}{2}, \frac{s-n-2}{2}) = \frac{s-(n+1)}{2}$.

A similar argument holds for state 1: $U^{t'}(1) = \max_{a \in A(1)} Q^{t'}(1, a) = Q^{t'}(1, a_1) \stackrel{\text{Theorem 22}}{\leq} Q^{t(1)+1}(1, a_1) = -1 + U^{t(1)}(succ(1, a_1)) \stackrel{\text{Theorem 22}}{\leq} -1 + U^t(2) \leq -1 + \frac{2-n}{2} = \frac{-n}{2} = \frac{1-(n+1)}{2}$.

A similar argument also holds for state n: $Q^{t(n)}(n, a_2) < Q^{t(n)}(n, a_1) \leq 0$. Since the Q-values are integers, $Q^{t'}(n, a_2) \stackrel{\text{Theorem 22}}{\leq} Q^{t(n)}(n, a_2) \leq -1$. $Q^{t'}(n, a_1) \stackrel{\text{Theorem 22}}{\leq} Q^{t(n)+1}(n, a_1) = -1 + U^{t(n)}(succ(n, a_1)) \stackrel{\text{Theorem 14}}{\leq} -1 + 0 = -1$. Then, $U^{t'}(n) = \max(Q^{t'}(n, a_1), Q^{t'}(n, a_2)) \leq \max(-1, -1) = -1 < -\frac{1}{2} = \frac{n-(n+1)}{2}$.

To summarize, $U^{t'}(s) \leq \frac{s-(n+1)}{2}$ for all $s \in S = \{1, 2, \ldots n+1\}$.

Theorem 59 *$Q(1, a_1) \leq \frac{1-n}{2}$ after termination of an undiscounted, admissible, and zero-initialized Q-learning algorithm in a reset state space of size $n \geq 1$ if ties are broken in favor of actions that lead to states with smaller numbers.*

Proof: $Q(1, a_1) = \max_{a \in A(1)} Q(1, a) = U(1) \stackrel{\text{Theorem 58}}{\leq} \frac{1-n}{2}$ after termination.

Theorem 60 *$Q(1, a_1) \leq \frac{2-n}{2}$ when an undiscounted, admissible, and zero-initialized Q-learning algorithm is the last time in state 1 of a reset state space of size $n \geq 2$ if ties are broken in favor of actions that lead to states with smaller numbers.*

Proof: Let t be the last step at which $s^t = 1$, and t' be the earliest step at which $s^{t'} = n-1$. The states $s \leq n-1$ form (together with their actions that do not leave

this set of states) a reset state space of size $n-1$. $Q^{t'}(1,a_1) \leq \frac{1-(n-1)}{2} = \frac{2-n}{2}$ according to Theorem 59. Since $Q^{t'}(n-1,a_1) = Q^{t'}(n-1,a_2) = 0$, $succ(n-1,a_2) = 1 < n = succ(n-1,a_1)$, and ties are broken in favor of actions that lead to states with smaller numbers, the agent chooses $a^{t'} = a_2$ for execution at step t' and is in state $s^{t'+1} = 1$ afterwards. Thus, $t' < t$ and $Q^t(1,a_1) \overset{\text{Theorem 22}}{\leq} Q^{t'}(1,a_1) \leq \frac{2-n}{2}$.

Theorem 61 *Figure 12 shows a domain for which an admissible, zero-initialized Q-learning algorithm needs at least* $\frac{(n+1)(n-1)(n-3)}{16} = 1/16n^3 - 3/16n^2 - 1/16n + 3/16$ *steps to reach the goal state and terminate (for odd* $n \geq 3$*) if ties are broken in favor of actions that lead to states with smaller numbers.*

Proof: The states $s \in \{\frac{n+1}{2}, \frac{n+3}{2}, \ldots, n\}$ form (together with their actions that do not leave this set of states) a reset state space of size $\frac{n+1}{2}$. $s_{start} = \frac{n+1}{2}$ is also the start state of the reset state space. Note that state $\frac{n+1}{2}$ separates the states that belong to the reset state space from the other states. Consider the trace that an admissible, zero-initialized Q-learning algorithm traverses in this reset state space if ties are broken in favor of actions that lead to states with smaller numbers. This state sequence is a subsequence of the state sequence that an admissible, zero-initialized Q-learning algorithm traverses in the whole domain (as shown in Figure 12) if ties are broken in favor of actions that lead to states with smaller numbers.

Let t' be the step when the algorithm has terminated in the domain from Figure 12 of size n (for odd $n \geq 3$) if ties are broken in favor of actions that lead to states with smaller numbers, i.e. $Q^{t'}(s,a)$ are the final Q-values. We assume (without loss of generality) that the Q-learning algorithm is undiscounted. Then, $Q^{t'}(\frac{n+1}{2}, a_1) \overset{\text{Theorem 59}}{\leq} \frac{1-\frac{n+1}{2}}{2} = \frac{1-n}{4}$, where $a_1 \in A(\frac{n+1}{2})$ is the action in state $\frac{n+1}{2}$ that decreases the goal distance. Similarly, let $t'' < t'$ be the last step at which $s^{t''} = \frac{n+1}{2}$ and $a^{t''} = a_1$. Then, $Q^{t''}(\frac{n+1}{2}, a_1) \overset{\text{Theorem 60}}{\leq} \frac{2-\frac{n+1}{2}}{2} = \frac{3-n}{4}$.

Now consider an arbitrary $a \in A(\frac{n+1}{2}) \setminus \{a_1\}$. $Q^{t''}(\frac{n+1}{2}, a) < Q^{t''}(\frac{n+1}{2}, a_1)$. (If $Q^{t''}(\frac{n+1}{2}, a) \geq Q^{t''}(\frac{n+1}{2}, a_1)$, then a would be preferred over a_1, since $succ(\frac{n+1}{2}, a) < \frac{n+1}{2} < \frac{n+3}{2} = succ(\frac{n+1}{2}, a_1)$ and ties are broken in favor of actions that lead to states with smaller numbers.) Since the Q-values are integers, $Q^{t''}(\frac{n+1}{2}, a) \leq Q^{t''}(\frac{n+1}{2}, a_1) - 1 \leq \frac{3-n}{4} - 1 < 0$. Thus, there is a step when a is executed in state $\frac{n+1}{2}$, since initially $Q(\frac{n+1}{2}, a) = 0$. Note that every path from a state smaller than $\frac{n+1}{2}$ to state n contains an execution of a_1 in state $\frac{n+1}{2}$, since since a_1 is the last action executed in $\frac{n+1}{2}$. Let $t(a) < t''$ be the last step when a is executed in $\frac{n+1}{2}$. For all $a' \in A(succ(\frac{n+1}{2}, a))$ it holds that $Q^{t'}(succ(\frac{n+1}{2}, a), a') \overset{\text{Theorem 22}}{\leq} Q^{t(a)}(succ(\frac{n+1}{2}, a), a') \leq \max_{a'' \in A(succ(\frac{n+1}{2}, a))} Q^{t(a)}(succ(\frac{n+1}{2}, a), a'') = U^{t(a)}(succ(\frac{n+1}{2}, a)) = 1 + Q^{t(a)+1}(\frac{n+1}{2}, a) = 1 + Q^{t'}(\frac{n+1}{2}, a) \overset{\text{Theorem 22}}{\leq} 1 + Q^{t''}(\frac{n+1}{2}, a) \leq Q^{t''}(\frac{n+1}{2}, a_1) \leq \frac{3-n}{4}$. Note that for every $s \in \{1, 2, \ldots, \frac{n-1}{2}\}$ both $|A(s)| = \frac{n-1}{2}$ and there exists an $a \in A(\frac{n+1}{2}) \setminus \{a_1\}$ such that $succ(\frac{n+1}{2}, a) = s$. Because the state space has no identity actions, $Q^{t'}(s,a) \leq 0$

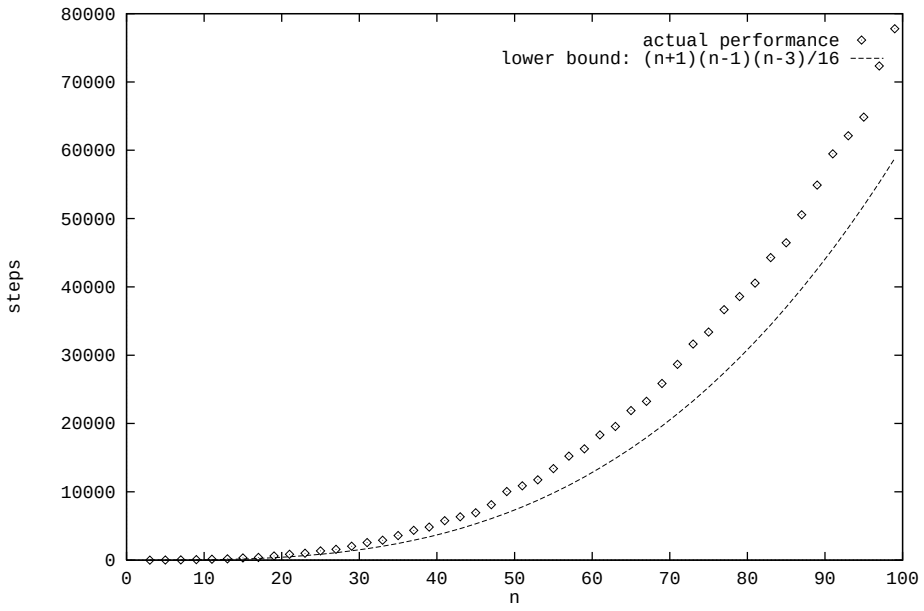

Figure 35: Run-time of an admissible, zero-initialized Q-learning algorithm for reaching the goal state in the domain shown in Figure 12 (for odd $n \in [3, 99]$) if ties are broken in favor of actions that lead to states with smaller numbers

---

for all $s \in S$ and $a \in A(s)$, $U^{t'}(n) = 0$ (since state n is a goal), and $|A(s)| \geq \frac{n-1}{2}$ for every state $s \leq \frac{n+1}{2}$, it holds that $t' \stackrel{\text{Theorem 26}}{=} U^{t'}(n) - \sum_{s \in S} \sum_{a \in A(s)} Q^{t'}(s,a) \geq 0 - \sum_{s=1}^{\frac{n+1}{2}} \sum_{a \in A(s)} Q^{t'}(s,a) \geq -\sum_{s=1}^{\frac{n+1}{2}} \sum_{a \in A(s)} \frac{3-n}{4} \geq \frac{n+1}{2} \frac{n-1}{2} \frac{n-3}{4} = \frac{(n+1)(n-1)(n-3)}{16} = 1/16n^3 - 3/16n^2 - 1/16 + 3/16$.

Theorem 2 and Theorem 61 together imply that the complexity of an admissible, zero-initialized Q-learning algorithm in the domain shown in Figure 12 is tight at $O(n^3)$. The actual number of steps needed by an admissible, zero-initialized Q-learning algorithm to reach the goal state and terminate in the domain shown in Figure 12 if ties are broken in favor of actions that lead to states with smaller numbers is not a smooth function of n. Since we ignored the Q-values of the states $s \in \{\frac{n+3}{2}, \frac{n+5}{2}, \ldots, n\}$ in the proof of Theorem 61, the lower bound $\frac{(n+1)(n-1)(n-3)}{16}$ is not tight. Figure 35 shows both the actual number of steps needed and the lower bound $\frac{(n+1)(n-1)(n-3)}{16}$ for odd $n \in [3, 99]$.

Theorem 62 *Figure 12 shows a domain for which the Q_{map}-learning algorithm needs at most $3/8n^2 + 3/2n - 23/8$ steps to reach the goal state and terminate (for odd $n \geq 3$).*

Proof: Note that state $\frac{n+1}{2}$ separates the states smaller than $\frac{n+1}{2}$ from the states larger than $\frac{n+1}{2}$. In the following, we assume that an arbitrary trace is given that the Q_{map}-learning algorithm has traversed when reaching the goal state from $s_{start} = \frac{n+1}{2}$. We analyze the trace in both subsets of states separately.

- Consider the states larger than or equal to $\frac{n+1}{2}$. We call the actions that decrease the goal distance action a_1, and the actions that lead the agent back to state $\frac{n+1}{2}$ action a_2. Consider any consecutive subsequence of the trace that starts and ends in state $\frac{n+1}{2}$ and otherwise contains only states larger than $\frac{n+1}{2}$. This subsequence must contain an action a_2 that is unexplored. (Since the start state and end state of the sequence are identical, it must contain at least one unexplored action. If it contains an unexplored action a_2, the statement is true. If it contains an unexplored action $a_1 \in A(s)$, then all states s' with $s' > s$ are still unexplored and thus no action has yet been executed in them. In particular, the action a_2 that the agent must execute in a state larger than s to reach state $\frac{n+1}{2}$ again is still unexplored.) If $a_2 \in A(s)$, then the length of the subsequence is $1+s-\frac{n+1}{2}$. There can be at most one such subsequence for every state $s \in \{\frac{n+3}{2}, \frac{n+5}{2}, \ldots, n-1\}$, since there is only one action a_2 in every such state. In addition, there is one subsequence of length $n - \frac{n+1}{2} = \frac{n-1}{2}$ that leads from state $\frac{n+1}{2}$ to the goal state n. Since these are all of the subsequences that contain states larger than $\frac{n+1}{2}$, the total of action executions in this part of the state space is at most $\sum_{s=\frac{n+3}{2}}^{n-1}(1+s-\frac{n+1}{2}) + \frac{n-1}{2} = 1/8n^2 + 1/2n - 13/8$.

- Now consider the states smaller than or equal to $\frac{n+1}{2}$. Delete first all states $\frac{n+1}{2}$ from the original trace that are followed by a state larger than $\frac{n+1}{2}$ and delete afterwards all states larger then $\frac{n+1}{2}$. (We have accounted for all action executions in these states in the paragraph above.) Note that zero-initialized Q-learning always executes an unexplored action in the current state if one is available, since the action execution step always executes the action with the largest Q-value and unexplored actions have a Q-value of 0 whereas explored actions have a Q-value smaller than 0. Consequently, Q_{map}-learning also executes unexplored actions if they are available. Since the states smaller than or equal to $\frac{n+1}{2}$ (together with the actions that do not leave this set of states) form a 1-step invertible state space, the shortened state sequence contains only executions of unexplored actions until all actions in state $\frac{n+1}{2}$ are explored. Then, the following behavior is repeated until all actions are explored: The agent executes an explored action that leads to a state s in which at least one action is still unexplored. Then, the agent executes only unexplored actions, until all actions in state s are explored, optionally followed by an explored action that leads to state $\frac{n+1}{2}$. To summarize, every of the $\frac{n+1}{2}\frac{n-1}{2}$ actions in this part of the state space is executed exactly once plus at most two actions for every state smaller than $\frac{n+1}{2}$, leading to a total number of action executions of $\frac{n+1}{2}\frac{n-1}{2} + 2\frac{n-1}{2} = 1/4n^2 + n - 5/4$.

In conclusion, the total number of action executions is bounded from above by $(1/8n^2 + 1/2n - 13/8) + (1/4n^2 + n - 5/4) = 3/8n^2 + 3/2n - 23/8$ no matter how ties among

actions that have the same Q-value are broken.

Note that every uninformed on-line search algorithm can need at least $1/2n^2 + 1/2n - 2$ steps to reach the goal state in a reset state space (shown in Figure 8) of size n (for $n \geq 2$):

```
for i := 2 to n
  for j := 1 to i
    print j
```

Now consider again the domain shown in Figure 12 and assume it has size n. As remarked earlier, the states $s \geq \frac{n+1}{2}$ form a reset state space of size $\frac{n+1}{2}$. Thus, every uninformed on-line search algorithm can need at least $1/2(\frac{n+1}{2})^2 + 1/2(\frac{n+1}{2}) - 2 = 1/8n^2 + 1/2n - 13/8$ steps (for odd $n \geq 3$). This fact and Theorem 62 together imply that the complexity of the Q_{map}-learning algorithm in the domain shown in Figure 12 is tight at $O(n^2)$.

B.5 Reaching a Goal State with Random Walks

Equations 4 allow one to determine the average number of steps required by a random walk to reach a goal state in a given domain, as outlined in the main text. Consider, for example, the reset state space shown in figure 8. The corresponding equations are

$$\begin{aligned} x_1 &= 1 + x_2 \\ x_s &= 1 + 0.5x_1 + 0.5x_{s+1} \qquad \text{for all } s \in \{2, 3, \ldots, n-1\} \\ x_n &= 0 \end{aligned}$$

They can be solved for $x_{s_{start}}$ as follows

$$\begin{aligned} x_{s_{start}} &= x_1 \\ &= 1 + x_2 \\ &= 1 + 1 + 0.5x_1 + 0.5x_3 \\ &= 1 + (1 + 0.5) + 0.5x_1(1 + 0.5) + 0.5^2 x_4 \\ &= 1 + (1 + 0.5 + 0.5^2) + 0.5x_1(1 + 0.5 + 0.5^2) + 0.5^3 x_5 \\ &= \ldots \\ &= 1 + \sum_{i=0}^{s-3} 0.5^i + 0.5x_1 \sum_{i=0}^{s-3} 0.5^i + 0.5^{s-2} x_s \qquad \text{for all } s \in \{2, 3, \ldots, n\} \\ &= \ldots \\ &= 1 + \sum_{i=0}^{n-3} 0.5^i + 0.5x_1 \sum_{i=0}^{n-3} 0.5^i + 0.5^{n-2} x_n \end{aligned}$$

$$
\begin{aligned}
&= 1 + \frac{1 - 0.5^{n-2}}{1 - 0.5} + 0.5x_1 \frac{1 - 0.5^{n-2}}{1 - 0.5} + 0.5^{n-2} \times 0 \\
&= 3 \times 2^{n-2} - 2
\end{aligned}
$$

The equations for the domains in figures 9, 17, and 18 can be derived in a similar way. In order to solve them, one usually uses generating functions, see for example [7] for a mathematical derivation and [37] for an application in the context of reinforcement learning. Since these derivations are long and cumbersome, we do not state them here, but make use of the fact that the solution is unique and show how to check a given solution by plugging it into the equations. Consider for example the one-dimensional gridworld shown in figure 18. The corresponding equations are

$$
\begin{aligned}
x_1 &= 1 + x_2 && (7)\\
x_s &= 1 + 0.5x_{s-1} + 0.5x_{s+1} \qquad \text{for all } s \in \{2, 3, \ldots, n-1\} && (8)\\
x_n &= 0 && (9)
\end{aligned}
$$

Consider the solution for $x_{s_{start}}$ stated in figure 18

$$x_{s_{start}} = x_1 = n^2 - 2n + 1$$

The solutions for the other variables x_s can be derived from this solution as follows

$$
\begin{aligned}
x_1 = 1 + x_2 = n^2 - 2n + 1 &\Rightarrow x_2 = n^2 - 2n \\
x_2 = 1 + 0.5x_1 + 0.5x_3 = n^2 - 2n &\Rightarrow x_3 = n^2 - 2n - 3 \\
x_3 = 1 + 0.5x_2 + 0.5x_4 = n^2 - 2n - 3 &\Rightarrow x_4 = n^2 - 2n - 8 \\
\ldots &\Rightarrow \ldots \\
x_{s-1} = 1 + 0.5x_{s-2} + 0.5x_s = n^2 - 2n - (s-1)^2 + 2(s-1) &\Rightarrow x_s = n^2 - 2n - s^2 + 2s
\end{aligned}
$$

for $s \in \{3, 4, \ldots, n\}$. Thus,

$$x_s = n^2 - 2n - s^2 + 2s \qquad \text{for all } s \in S = \{1, 2, \ldots, n\}$$

The x_s for $s \in S$ were constructed to satisfy Equation 7 and Equations 8. In order to prove that the x_s are indeed a solution of the set of equations, we are left to verify Equation 9 and indeed

$$x_n = n^2 - 2n - n^2 + 2n = 0$$

C Value-Iteration and Q-Learning

In the following, we use the transformation rule, terminology, and definitions of symbols introduced in Chapter 6.1 to show how results about value-iteration can be transferred to Q-learning and vice versa.

The time superscripts used in this chapter refer to the values of the variables immediately before the action execution step of the Q-learning or value-iteration algorithm, i.e. line 4, of step t.

Theorem 63 *For all $s \in S \backslash G$ and $a \in A(s)$, it holds that $\widetilde{gd}(s_{s,a}) = gd(succ(s,a))+1$.*

Proof: Assume that we execute action $a_y \in A(s_y)$ in state $s_y \in S$ at time $1 \leq y \leq x := gd(succ(s,a))$ while following a shortest path from state $s_1 := succ(s,a)$ to the closest goal state $s_{x+1} := succ(s_x, a_x) \in G$. Let $a_{x+1} \in A(s_{x+1})$ be an arbitrary action in s_{x+1}. Then, the state sequence $s_{s,a}, s_{s_1,a_1}, s_{s_2,a_2}, \ldots, s_{s_x,a_x}, s_{s_{x+1},a_{x+1}}$ of states in $\widetilde{S}$ describes a path from state $s_{s,a}$ to the goal state $s_{s_{x+1},a_{x+1}} \in \widetilde{G}$ of length $x+1$. Thus, $\widetilde{gd}(s_{s,a}) \leq x+1 = gd(succ(s,a))+1$.

Conversely, assume that we execute action $\widetilde{a}_y \in \widetilde{A}(\widetilde{s}_y)$ in state $\widetilde{s}_y = s_{s_y,a_y} \in \widetilde{S}$ at time $1 \leq y \leq x := \widetilde{gd}(s_{s,a})$ while following a shortest path from state $\widetilde{s}_1 := s_{s,a}$ to the closest goal state $\widetilde{s}_{x+1} := \widetilde{succ}(\widetilde{s}_x, \widetilde{a}_x) \in \widetilde{G}$. Then, the state sequence $s_2, s_3, \ldots, s_{x+1}$ of states in S describes a path from $s_2 = succ(s,a)$ to the goal state $s_{x+1} \in G$ of length $x-1$, which cannot be shorter than $gd(succ(s,a))$. Thus, $\widetilde{gd}(s_{s,a}) = x \geq gd(succ(s,a))+1$.

Put together, it follows that $\widetilde{gd}(s_{s,a}) = gd(succ(s,a))+1$.

Theorem 64 $d \leq \widetilde{d} \leq d+1$.

Proof: We prove $d \leq \widetilde{d}$ and $\widetilde{d} \leq d+1$ separately.

Consider two arbitrary states $s_{s,a}, s_{s',a'} \in \widetilde{S}$. We show that $\widetilde{d}(s_{s,a}, s_{s',a'}) \leq d+1$. Assume that we execute action $a_y \in A(s_y)$ in state $s_y \in S$ at time $1 \leq y \leq x := d(succ(s,a), s') \leq d$ while following a shortest path from state $s_1 := succ(s,a)$ to state $s_{x+1} := succ(s_x, a_x) := s'$. Then, the state sequence $s_{s,a}, s_{s_1,a_1}, s_{s_2,a_2}, \ldots, s_{s_x,a_x}, s_{s',a'}$ of states in $\widetilde{S}$ describes a path from state $s_{s,a}$ to state $s_{s',a'}$ of length $x+1$. Thus, $\widetilde{d}(s_{s,a}, s_{s',a'}) \leq x+1 \leq d+1$ and $\widetilde{d} = \max_{\widetilde{s},\widetilde{s}' \in \widetilde{S}} \widetilde{d}(\widetilde{s},\widetilde{s}') = \max_{\widetilde{s},\widetilde{s}' \in \widetilde{S}} (d+1) \leq d+1$.

Next, consider two arbitrary states $s, s' \in S$ together with two arbitrary actions $a \in A(s)$ and $a' \in A(s')$. We show that $\widetilde{d} \leq d(s,s')$. Assume that we execute action $\widetilde{a}_y \in \widetilde{A}(\widetilde{s}_y)$ in state $\widetilde{s}_y = s_{s_y,a_y} \in \widetilde{S}$ at time $1 \leq y \leq x := \widetilde{d}(s_{s,a}, s_{s',a'}) \leq \widetilde{d}$ while following a shortest path from state $\widetilde{s}_1 := s_{s,a}$ to state $\widetilde{s}_{x+1} := \widetilde{succ}(\widetilde{s}_x, \widetilde{a}_x) := s_{s',a'}$. Then, the state sequence $s_1, s_2, \ldots, s_{x+1}$ of states in S describes a path from state $s_1 = s$ to state $s_{x+1} = s'$ of length x, which cannot be shorter than $d(s,s')$. Thus $\widetilde{d} \geq x \geq d(s,s')$ and $d = \max_{s,s' \in S} d(s,s') \leq \max_{s,s' \in S} \widetilde{d} = \widetilde{d}$.

Figures 36 and 37 show that indeed both $\widetilde{d} = d$ and $\widetilde{d} = d+1$ are possible. Since $\widetilde{d} \leq d+1$, the transformed domain is guaranteed to be strongly connected.

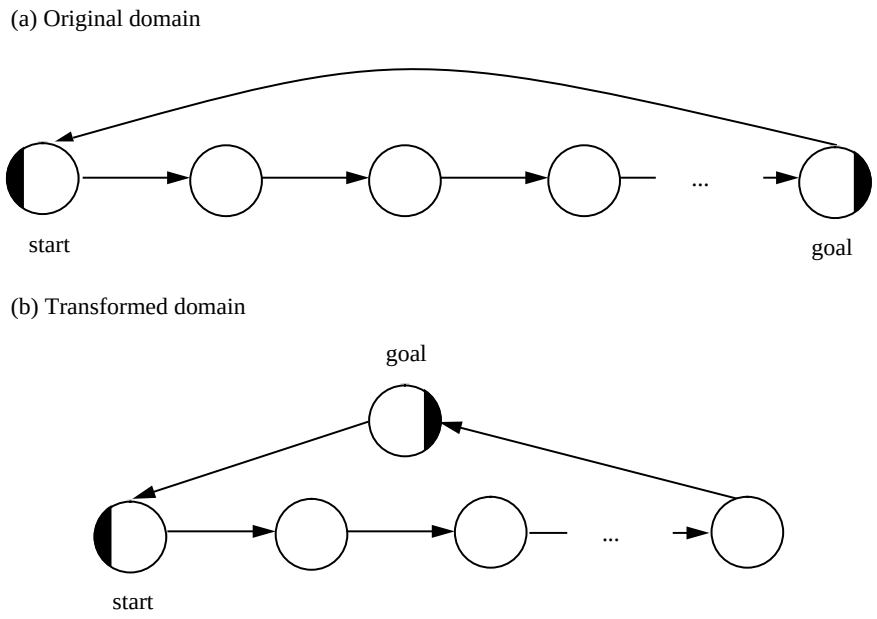

Figure 36: A domain with $d = n - 1$ and its transformation with $\widetilde{d} = d$

Theorem 65 *Undiscounted, admissible value-iteration with action-penalty representation behaves in the transformed domain identically to undiscounted, admissible Q-learning with action-penalty representation in the original domain if the original domain has no identity actions, initially $Q(s,a) = U(s_{s,a})$ for all $s \in S$ and $a \in A(s)$, and ties are broken in the same way. Also, $U^t(s_{s,a}) = Q^t(s,a)$ for all $s \in S$, $a \in A(s)$, and steps t (until termination).*

Proof: We prove by induction on t that $U^t(s_{s,a}) = s_{s^1,a^1}$ for all $s \in S$, $a \in A(s)$, and steps t (until termination). At the same time, we prove that $\widetilde{s}^t = s_{s^t,a^t}$ for all steps t (until termination).

- Initially, $\widetilde{s}^1 = \widetilde{s}_{start} = s_{s_{start},argmax_{a \in A(s_{start})} Q(s_{start},a)} = Q(s^1,a^1)$ if ties are broken in the same way. Also, $U^1(s_{s,a}) = Q^1(s,a)$ for all $s \in S$ and $a \in A(s)$ per definition.
- Assume that the two statements hold for step t.

  First, we prove that $\widetilde{a}^t = a^{t+1}$, i.e. both algorithms select the same actions.

$$\begin{aligned} \widetilde{a}^t &= \text{argmax}_{\widetilde{a} \in \widetilde{A}(\widetilde{s}^t)} U^t(\widetilde{succ}(\widetilde{s}^t, \widetilde{a})) \\ &= \text{argmax}_{\widetilde{a} \in \widetilde{A}(s_{s^t,a^t})} U^t(\widetilde{succ}(s_{s^t,a^t}, \widetilde{a})) \end{aligned}$$

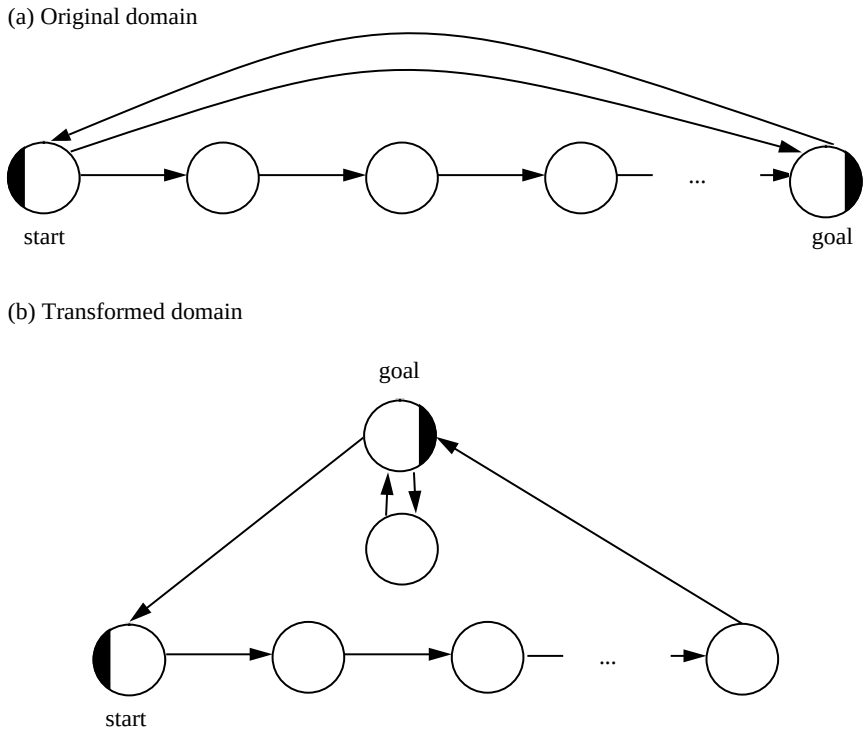

Figure 37: A domain with $d = n - 1$ and its transformation with $\widetilde{d} = d + 1$

$$
\begin{aligned}
&= \text{argmax}_{\widetilde{a} \in \widetilde{A}(s_{s^t,a^t})} U^t(s_{succ(s^t,a^t),\widetilde{a}}) \\
&= \text{argmax}_{a' \in A(succ(s^t,a^t))} Q^t(succ(s^t,a^t),a') \\
&= \text{argmax}_{a' \in A(s^{t+1})} Q^t(s^{t+1},a') \\
&= \text{argmax}_{a' \in A(s^{t+1})} Q^{t+1}(s^{t+1},a') \\
&= a^{t+1}
\end{aligned}
$$

if ties are broken in the same way. Note that $Q(s^t, a^t)$ is the only Q-value of the original domain that changes from step t to step $t + 1$. Since the domain has no identity actions, it holds that $s^t \neq s^{t+1}$ and therefore $Q^t(s^{t+1}, a') = Q^{t+1}(s^{t+1}, a')$ for all $a' \in A(s^{t+1})$.

Next, we prove that $U^{t+1}(\widetilde{s}^t) = Q^{t+1}(s^t, a^t)$.

$$
\begin{aligned}
U^{t+1}(\widetilde{s}^t) &= \max_{\widetilde{a} \in \widetilde{A}(\widetilde{s}^t)} (-1 + U^t(\widetilde{succ}(\widetilde{s}^t, \widetilde{a}))) \\
&= \max_{\widetilde{a} \in \widetilde{A}(s_{s^t,a^t})} (-1 + U^t(\widetilde{succ}(s_{s^t,a^t}, \widetilde{a}))) \\
&= \max_{\widetilde{a} \in \widetilde{A}(s_{s^t,a^t})} (-1 + U^t(s_{succ(s^t,a^t),\widetilde{a}}))
\end{aligned}
$$

$$
\begin{aligned}
&= -1 + \max_{a' \in A(succ(s^t,a^t))} Q^t(succ(s^t,a^t),a') \\
&= -1 + U^t(succ(s^t,a^t)) \\
&= Q^{t+1}(s^t,a^t)
\end{aligned}
$$

Since the only U-value that changes from step t to step $t+1$ in the transformed domain is $U(\widetilde{s}^t) = U(s_{s^t,a^t})$, the only Q-value that changes during the same time span in the original domain is $Q(s^t,a^t)$, and $U^t(s_{s,a}) = Q^t(s,a)$ for all $s \in S$ and $a \in A(s)$, it follows that $U^{t+1}(s_{s,a}) = Q^{t+1}(s,a)$ for all $s \in S$ and $a \in A(s)$ as well.

Finally, we prove that $\widetilde{s}^{t+1} = s_{s^{t+1},a^{t+1}}$. $\widetilde{s}^{t+1} = \widetilde{succ}(\widetilde{s}^t,\widetilde{a}^t) = \widetilde{succ}(s_{s^t,a^t},\widetilde{a}^t) = s_{succ(s^t,a^t),a^{t+1}} = s_{s^{t+1},a^{t+1}}$.

Q-learning terminates at the earliest step t at which $s^t \in G$. Thus, $s^{t'} \notin G$ for $t' < t$. Since $\widetilde{s}^{t'} = s_{s^{t'},a^{t'}}$ for all steps t', it follows that $\widetilde{s}^{t'} \notin \widetilde{G}$ for $t' < t$ and $\widetilde{s}^t \in \widetilde{G}$. Since the value-iteration algorithm terminates at the earliest time at which its current state is a goal state, it terminates at step t.

Theorem 66 *The U-values of the transformed domain are consistent iff the Q-values of the original domain are consistent.*

Proof: Note that $\widetilde{s} = s_{s,a} \in \widetilde{G}$ iff $s \in G$, for all $s \in S$ and $a \in A(s)$. Also $U(s_{s,a}) = Q(s,a)$ for all $s \in S$ and $a \in A(s)$ according to Theorem 65. The U-values are consistent iff, for all $s_{s,a} \in \widetilde{G}$, $U(s_{s,a}) = 0$, and, for all $s_{s,a} \in \widetilde{S} \setminus \widetilde{G}$, $\max_{\widetilde{a} \in \widetilde{A}(s_{s,a})}(-1 + U(\widetilde{succ}(s_{s,a},\widetilde{a}))) \leq U(s_{s,a}) \leq 0$.

$U(s_{s,a}) = 0$ for all $s_{s,a} \in \widetilde{G}$ is equivalent to $Q(s,a) = U(s_{s,a}) = 0$ for all $s \in G$.

$\max_{\widetilde{a} \in \widetilde{A}(s_{s,a})}(-1 + U(\widetilde{succ}(s_{s,a},\widetilde{a}))) \leq U(s_{s,a}) \leq 0$ for all $s_{s,a} \in \widetilde{S} \setminus \widetilde{G}$ is equivalent to the following inequalities

$$
\begin{aligned}
&\max_{\widetilde{a} \in \widetilde{A}(s_{s,a})} (-1 + U(s_{succ(s,a),\widetilde{a}})) \leq U(s_{s,a}) \leq 0 \\
&-1 + \max_{a' \in A(succ(s,a))} Q(succ(s,a),a') \leq Q(s,a) \leq 0 \\
&-1 + U(succ(s,a)) \leq Q(s,a) \leq 0
\end{aligned}
$$

for all $s \in S \setminus G$.

Put together, it follows that the U-values of the transformed domain are consistent iff the Q-values of the original domain are consistent.

Theorem 67 *The U-values of the transformed domain are admissible iff the Q-values of the original domain are admissible.*

Proof: Note that $\widetilde{s} = s_{s,a} \in \widetilde{G}$ iff $s \in G$, for all $s \in S$ and $a \in A(s)$. Also $U(s_{s,a}) = Q(s,a)$ for all $s \in S$ and $a \in A(s)$ according to Theorem 65. The U-values are admissible iff, for all $\widetilde{s} = s_{s,a} \in \widetilde{S}$, $-\widetilde{gd}(s_{s,a}) \leq U(s_{s,a}) \leq 0$, which is equivalent to, for all $s_{s,a} \in \widetilde{G}$, $U(s_{s,a}) = 0$, and, for all $s_{s,a} \in \widetilde{S} \setminus \widetilde{G}$, $-\widetilde{gd}(s_{s,a}) \leq U(s_{s,a}) \leq 0$.

$U(s_{s,a}) = 0$ for all $s_{s,a} \in \widetilde{G}$ is equivalent to $Q(s,a) = U(s_{s,a}) = 0$ for all $s \in G$.

$-\widetilde{gd}(s_{s,a}) \leq U(s_{s,a}) \leq 0$ for all $s_{s,a} \in \widetilde{S} \setminus \widetilde{G}$ is equivalent to $-1 - gd(succ(s,a)) = -\widetilde{gd}(s_{s,a}) \leq U(s_{s,a}) = Q(s,a) \leq 0$ for all $s \in S \setminus G$ according to Theorem 63.

Put together, it follows that the U-values of the transformed domain are admissible iff the Q-values of the original domain are admissible.
