Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331
tgd@cs.orst.edu

August 24, 1997

Abstract

This paper describes the MAXQ method for hierarchical reinforcement learning based on a hierarchical decomposition of the value function and derives conditions under which the MAXQ decomposition can represent the optimal value function. We show that for certain execution models, the MAXQ decomposition will produce better policies than Feudal Q learning.

1 Introduction

Hierarchical approaches to reinforcement learning (RL) problems promise many benefits: (a) improved exploration (because exploration can take "big steps" at high levels of abstraction), (b) learning from fewer trials (because fewer parameters must be learned), and (c) faster learning for new problems (because subtasks learned on previous problems can be re-used). Previous methods for hierarchical RL (Singh, 1992; Kaelbling, 1993; Dayan & Hinton, 1993) were tailored to specific tasks. This paper introduces a new method that unifies and extends these previous methods so that they can be applied to arbitrary hierarchical RL problems.

We illustrate the MAXQ method using the simple robot navigation task introduced by Dayan and Hinton (1993) and shown in Figure 1. In this task, an 8x8 grid world is progressively abstracted so that 2x2 sets of squares at each level are represented by a single square at the next higher level of abstraction. Each level can be thought of as an agent that must obey the commands of the level above it (its feudal lord) and that can give commands to the level below it (its vassals). At level 3, there are four primitive operators, North, South, East, and West, each of which moves one square in the designated direction and has a cost of 1 unit. In each abstract state, there are five possible operators: North, South, East, West, and *. An abstract operator at one level becomes a "goal" at the next level. For example, the North operator at state 1-(1,1) in level 1 requires level 2 to take actions to move to a square whose abstract location is 1-(1,2). The overall task is to find minimum-cost paths between every possible pair of squares.

When the current square and the goal square are the same at an abstract level (e.g., if the current square is 3-(6,6) and the goal square is 3-(8,8), then at level 1, both squares are at 1-(2,2)), then the * operator is the only applicable operator. This operator commands the next lower level to search for the goal square at its level. The next lower level can infer that the goal square is in one of the four squares that make up the abstract state at the upper level (e.g., 2-(3,3), 2-(3,4), 2-(4,3) and 2-(4,4)). The * operator is therefore mapped to one of the four subgoals ne, se, sw,

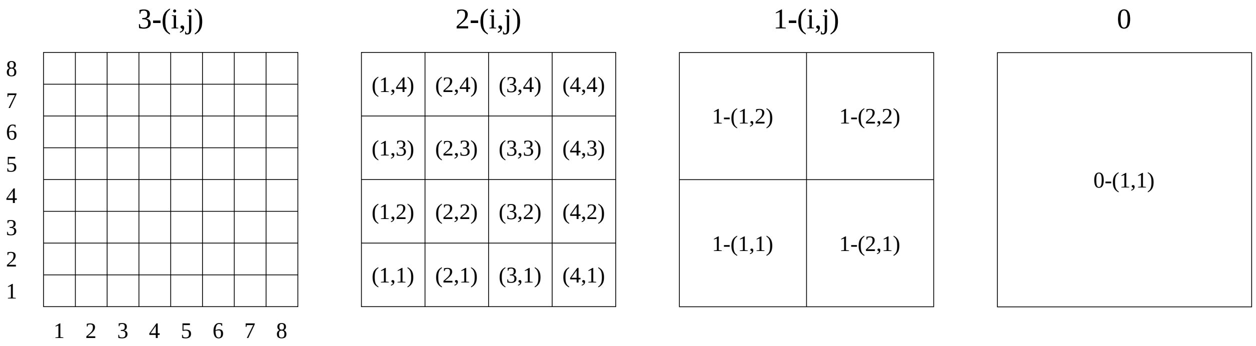

Figure 1: An 8x8 navigation task studied by Dayan and Hinton. The left-most grid is the actual task. The other three grids are progressive abstractions.

and nw. These subgoals are successfully achieved at a level if the lower levels are able to change the abstract state to be the corresponding square. In our example, the current square at level 2 is 2-(3,3) and the goal square is at 2-(4,4), so the * is converted to a ne subgoal at level 2.

In Feudal Q learning, each level i of the hierarchy learns a Q function of the form $Q_i(s, a, g)$, where s is the abstract state at level i, a is an action at level i, and g is the subgoal at level i (dictated by the feudal lord at level $i - 1$). If the lord chose the * operator, then g is the mapped subgoal. The Q value gives the expected reward of achieving goal g starting in state s and applying action a:

$$Q_i(s, a, g) = \sum_{s'} P(s'|s, a)[R_i(s'|s, a) + \min_u Q_i(s', u, g)],$$

where $P(s'|s, a)$ is the probability of moving to state s' when action a is executed in state s. The reward at level i, $R_i(s'|s, a)$, is the cost incurred by the lower levels in implementing the action a.

The Feudal Q hierarchy is executed using a *commitment* regime similar to procedure call and return. At level 0, only the operator * is applicable, so it is always executed. At all other levels, the action u with the highest Q value is chosen and executed until the goal g is achieved or until the abstract state at level $i - 1$ changes.

2 The MAXQ Value Function Decomposition

Figure 2 shows a MAXQ graph for the 8x8 navigation task. The graph contains three kinds of nodes—MAX nodes, Q nodes, and feature nodes. Of these, only the Q nodes are learned. MAX nodes are drawn as triangles. MAX nodes with no children represent the primitive actions in the domain (e.g., N, S, E, and W at level 3). MAX nodes with children represent subtasks (e.g., the $f_3(g_2)$ node is the subtask of making moves at level 3 to achieve the subgoal g_2 dictated by level 2).

The children of a MAX node are oval-shaped Q nodes. Each Q node holds the Q value of an action that can be taken to achieve the task of its parent MAX node. For example, the $S2(g_1)$ node computes the Q value of taking the South action at level 2 when the level 1 parent chose the goal g_1. This is the expected cost of performing the South action and then following the current level 2 policy until g_1 is achieved. Every Q node has a (possibly-shared) child MAX node that implements its action. In the case of $S2$, this is the $f_3(g_2)$ node with $g_2 =$ S. Each Q node also has a set of rectangular feature nodes that point to it. These provide the features of the current state

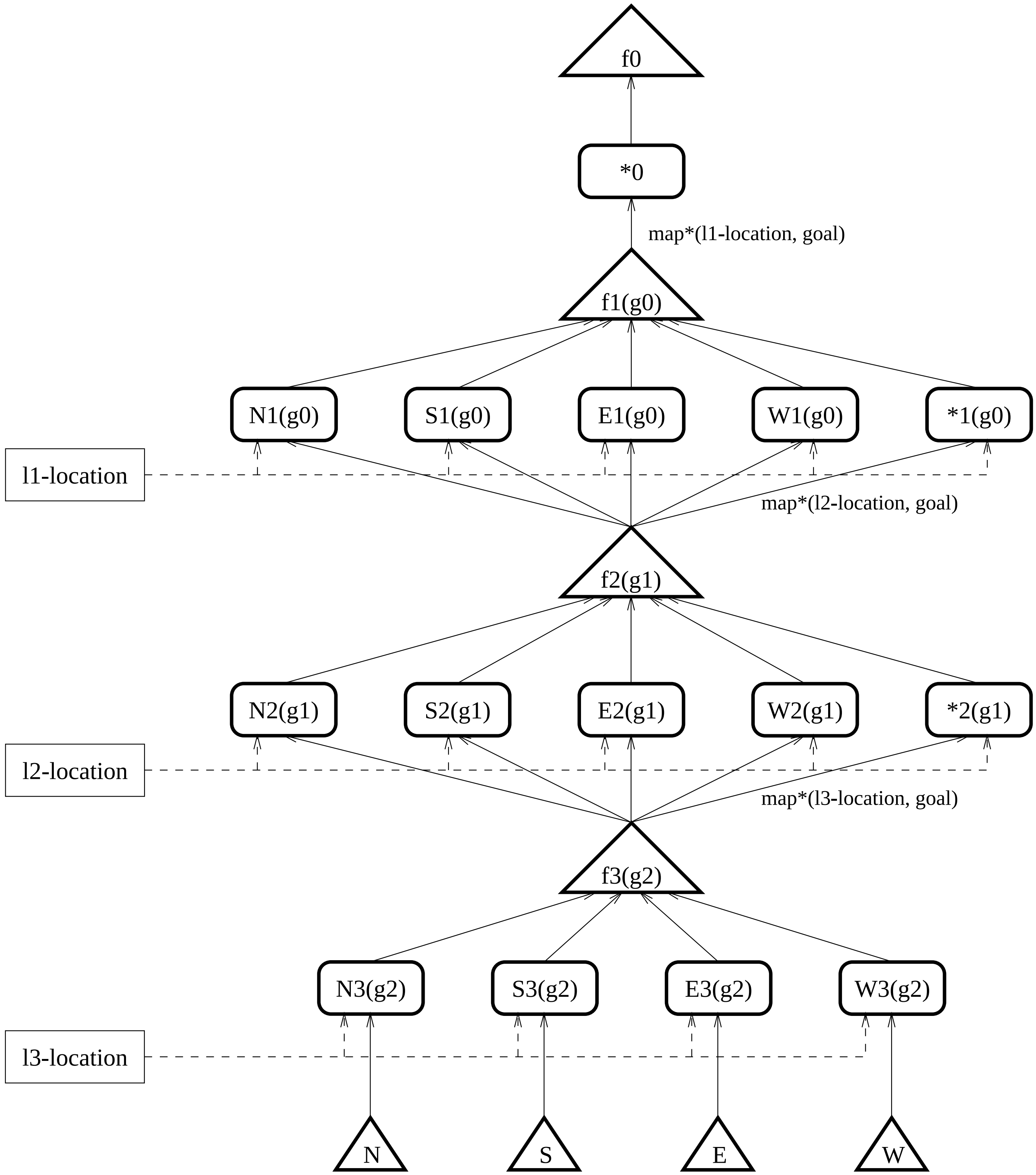

Figure 2: A MAXQ graph for the navigation task from Figure 1

that are relevant to representing the Q value of that state. For $S2$, the features are the abstract row and column of the current location at level 2.

Formally, the MAXQ decomposition can be described as follows. Define an overall MDP M over a set of states S, with primitive actions U^p, state transition probabilities $P(s'|s,u)$ for $u \in U^p$, and reward function $R(s'|s,u)$. Let U^c be the set of composite MAX nodes in the MAXQ graph, and define $U = U^c \cup U^p$ to be the set of all MAX nodes in the graph. To simplify notation, we will assume there are no parameterized MAX or Q nodes in the graph—these can be added in the obvious way. For each composite MAX node, $MAX_u, u \in U^c$, we will define a Markov decision problem over a set of states $S_u \subseteq S$, a set of terminal states $T_u \subset S_u$, and a set of actions $U_u \subset U$.

Now consider a fixed policy π for the hierarchy that consists of a collection of individual policies, π_u, for each MAX node $u \in U^c$. The root node MAX_0 chooses composite actions from U_0 according to π_0. Those composite actions in turn choose child actions recursively until primitive actions are chosen at the leaves of the graph. Define $MAX_u(s)$ to be the expected cumulative reward of following policy π_u starting in state s until some state in T_u is reached. The cumulative reward is the sum of $R(s'|s,v)$ of each primitive action $v \in U^p$ that is performed while executing π_u. Because of the hierarchical structure of the graph, we can write this as

$$MAX_u(s) = MAX_v(s) + \sum_{s'} P(s'|s,v) MAX_u(s'),$$

where $v = \pi_u(s)$ is the child action chosen according to policy π_u. In other words, $MAX_v(s)$ plays the role of the immediate reward function for taking action v as part of task u.

Now we adopt the state-action (Q function) representation of MAX_u:

$$\begin{aligned}
Q_{u,v}(s) &= MAX_v(s) + C_{u,v}(s) &(1)\\
C_{u,v}(s) &= \sum_{s'} P(s'|s,v) MAX_u(s') &(2)\\
MAX_u(s) &= \begin{cases} Q_{u,\pi_u(s)}(s) & \text{if } u \text{ is composite} \\ \sum_{s'} P(s'|s,u) R(s'|s,u) & \text{if } u \text{ is primitive} \end{cases} &(3)
\end{aligned}$$

Here, we have introduced $C_{u,v}$ to be the cost-to-go function. It represents the expected cost of following policy π_u (*after* executing action v) until a terminal state in T_u is entered. It is this cost-to-go function that is actually learned and stored in each Q node of the MAXQ graph. Note, in Equation (3), that the MAX node simply returns the Q value of the action chosen by π_u.

We will say that a MAXQ graph is a *full-state* graph, if separate $C_{u,v}$ values are stored for each state $s \in S_u$. In most applications, including Figure 1, it will be desirable to introduce an abstraction function $X_u(s)$ that will provide a set of features that abstract essential information from the state. Each Q node will then store the function $C_{u,v}(X_u(s))$, with one value for each distinct abstract state $X_u(s)$.

It is easy to prove the following theorem by expanding Equations (1-3):

Theorem 1 *Let $\pi = \{\pi_u | u \in U^c\}$ be the hierarchical policy defined by a full-state MAXQ graph and let MAX_0 be the root node of the graph. Then $MAX_0(s)$ is equal to the expected cumulative reward of following policy π in state s.*

Under what conditions will an abstract-state MAXQ graph be able to exactly represent the value function? The following theorem establishes one condition:

Theorem 2 *For all MAX nodes u and actions v, let $Result(s,v) = \{s' | P(s'|s,v) > 0\}$ be the set of states that can result from applying abstract action v in state s. Let $MAX_0(s)$ be the value* *function computed by a full-state MAXQ graph, and $\widehat{MAX}_0(s)$ be the value function computed by the corresponding abstract-state MAXQ graph with abstraction functions $\{X_u\}$. Then $MAX_0(s) = \widehat{MAX}_0(s)$ if for all nodes u, actions v, and $s_1, s_2 \in Result(s, v)$ such that $X_u(s_1) = X_u(s_2)$, $MAX_u(s_1) = MAX_u(s_2)$.*

In other words, if an abstraction function X_u treats a pair of result states s_1 and s_2 as identical, then their un-abstracted cost-to-go values must be equal. Otherwise, the value function cannot be properly represented. One case that satisfies this condition is when each abstract action has a single result state.

3 Learning in the MAXQ Hierarchy

While following the fixed hierarchical policy π, the $C_{u,v}$ values can be learned by the SARSA rule:

$$C_{u,v}(s) := (1-\alpha)C_{u,v}(s) + \alpha MAX_u(s'), \tag{4}$$

where s' is the state that results from executing action v in state s. However, if instead of using the fixed policy π, we use the resulting C values to select actions (i.e., by choosing $\pi_u(s) = \operatorname{argmax}_v Q_{u,v}(s)$), we will not achieve the intended result. The problem is that in most hierarchical tasks, subtasks have both desired and undesired terminal states. For example, in Figure 1, if we are in state 2-(1,1) and our parent asks us to move East, then the desired terminal states are 2-(3,1) and 2-(3,2). However, states 2-(1,3) and 2-(2,3) are also terminal states, but they are *not* desired. If we apply Equation 4, it will learn the optimal policy for reaching *any* of the terminal states, not just the desired ones. To enforce the intended semantics of abstract actions, Dayan and Hinton provided a penalty of 10 points for entering an undesired terminal state. This has the proper effect, but in the MAXQ hierarchy, it causes the value function computed by the entire hierarchy to be incorrect, because it incorporates the probability of receiving these terminal state penalties.

A better method is to define, for each MAX node MAX_u, a parallel Markov decision problem $\tilde{M}_u$ with states S_u, actions U_u, and a reward function $\tilde{R}_u$ that is zero except for undesired terminal states, where it provides a large penalty. For each of these parallel problems, we can apply standard Q learning to learn $\tilde{C}$ based on $\tilde{R}_u$:

$$\tilde{C}_{u,v}(s) := (1-\alpha)\tilde{C}_{u,v}(s) + \alpha[\tilde{R}_u(s') + \max_a \tilde{Q}_{u,a}(s')] \tag{5}$$

where

$$\tilde{Q}_{u,v}(s) = MAX_v(s) + \tilde{C}_{u,v}(s). \tag{6}$$

Note that $MAX_v(s)$ is the true expected cost of taking action v. The policy can now be defined by $\pi_u(s) = \operatorname{argmax}_v \tilde{Q}_{u,v}(s)$.

The resulting MAXQ hierarchy exhibits what we call "Bureaucratic Q Learning". Each MAX node is like a manager in a bureaucracy. To decide how to act, each "manager" consults her direct "employees" and says "If I gave you responsibility for taking the next action, how much do you estimate it would cost?" Each "employee" consults her "employees" recursively until the "workers" at the bottom of the hierarchy indicate the expected cost of alternative primitive actions. This information then percolates back up the hierarchy, where each "manager" u chooses her best "employee" $v = \pi_u(s)$ and adds in her own estimate $C_{u,v}$ of the remaining cost-to-go. The result is that at the root, there is some "best path" of "employees," $(u_1, u_2, \ldots, u_n)$ who have computed the value:

$$MAX_0(s) = C_{0,u_1}(s) + C_{u_1,u_2}(s) + \ldots + C_{u_{n-1},u_n}(s) + MAX_{u_n}(s). \tag{7}$$

To summarize: learning and selection of actions are performed at each node using the parallel MDP, $\tilde{M}_u$. However, the values passed up the hierarchy are based on the original MDP, M_u.

There are two possible execution models for the MAXQ hierarchy: *committed* and *polling*. In committed execution, the root node MAX_0 chooses the action a whose $\tilde{Q}_{0,a}$ value is maximized. It then commits to executing action a. That is, the subtask MAX_a is executed until it enters one of its terminal states T_a, at which point control returns to MAX_0. This is essentially the execution model employed by Dayan and Hinton, except that for Feudal Q, the values $\tilde{Q}_{0,a}$ are computed without consulting the "employees".

In the polling execution model, MAX_0 again computes the action a that maximizes $\tilde{Q}_{0,a}$, but in addition, it records the entire downward path $(a, u_1, \ldots, u_n)$ that is responsible for this value. The primitive action u_n is then executed, and the entire computation is repeated. Polling execution was employed by Kaelbling (1993) in her HDG method.

Upon reflection, we can see that the policy computed by polling execution is equivalent to the policy improvement step of the policy improvement dynamic programming algorithm. In policy improvement,

$$\pi^{t+1}(s) := \operatorname*{argmax}_a Q^t(s,a)$$

In contrast, the policy computed by committed execution is a less-than-complete policy improvement step

$$\pi^{t+1}(s) := \operatorname*{argmax}_{a \in U_u^*} Q_u^t(s,a),$$

where u is the index of the MAX node MAX_u that is currently in control of choosing the next action, and U_u^* is the set of all primitive actions that are leaf-node descendents of u in the hierarchy. Because the choice of possible actions is limited to those in U_u^*, committed execution may miss good actions a that polling execution will identify. This informally proves the following:

Theorem 3 *For all states s, the value of the policy computed by polling execution of the MAXQ hierarchy is $\geq$ the value of the policy computed by committed execution.*

A consequence of this theorem is that the MAXQ hierarchy can find optimal policies for problems where Feudal Q learning cannot. Figure 3 shows an instance of the Dayan/Hinton navigation task demonstrating this. In the lower left 4x4 region, Feudal Q chooses the abstract action North at level 1, because this is the best action for 13 of the 16 squares in this region, and Feudal Q "averages" over all 16 squares. However, this means that for states 3-(3,1), 3-(4,1), and 3-(4,2), it will take a sub-optimal action. The optimal action in these states is to move south and east through the doorway between 3-(4,1) and 3-(5,1). With the MAXQ representation, the MAX node $f_1(g_0)$ will choose the East action in these states, because its "employees" report that it is easier to achieve than a North action and the cost-to-go in the resulting states is the same. This example shows one way in which "bureaucracies" are better than "autocracies".

4 Concluding Remarks

The MAXQ decomposition generalizes previous work in hierarchical reinforcement learning for MDP's by Singh (1992) and Kaelbling (1993), as well as Dayan and Hinton (1993). We have implemented each of the tasks discussed by these authors using MAXQ graphs. In addition, we are applying the graphs to a complex real-time air traffic control simulation. We hope that others will find this formalism useful for designing solutions to complex MDPs.

Acknowledgements. The author thanks Prasad Tadepalli, Eric Chown, Bill Langford, and Rich Sutton for helpful discussions.

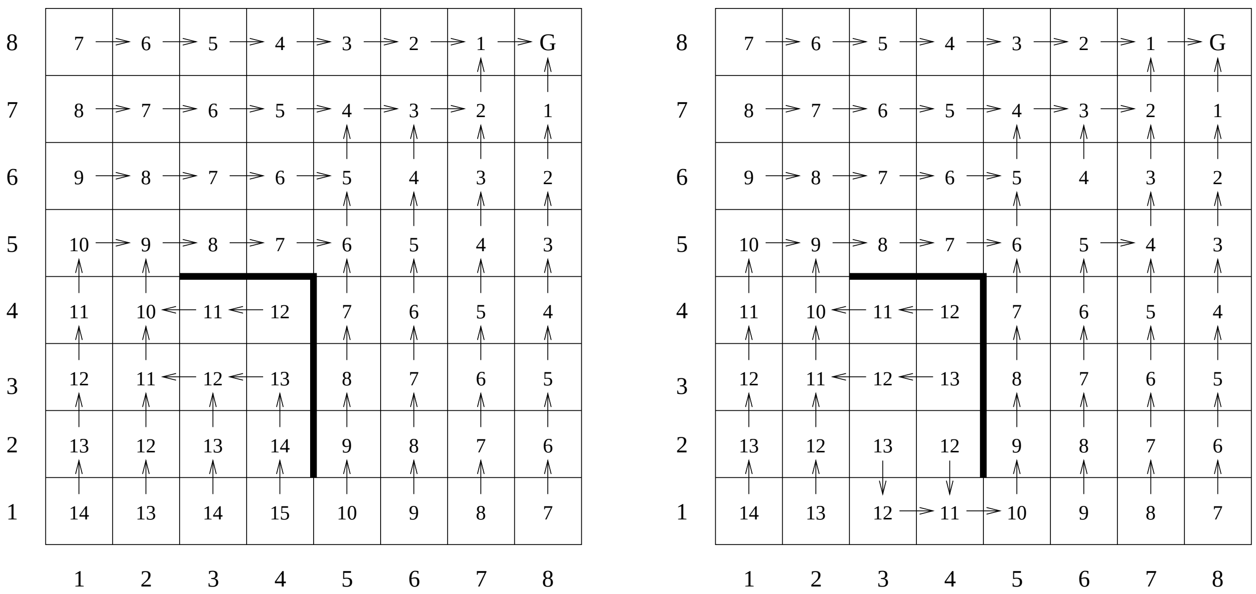

Figure 3: A maze task with a wall. Left: Feudal Q policy and value function. Right: MAXQ policy and value function.

References

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. In *Adv. in Neural Info. Proc. Sys., 5*. (pp. 271–278) Morgan Kaufmann, San Francisco, CA.

Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 167–173) San Francisco, CA. Morgan Kaufmann.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, *8*, 323–339.