Compositional Models for Reinforcement Learning

Nicholas K. Jong and Peter Stone

The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712
United States
{nkj,pstone}@cs.utexas.edu

Abstract. Innovations such as optimistic exploration, function approximation, and hierarchical decomposition have helped scale reinforcement learning to more complex environments, but these three ideas have rarely been studied together. This paper develops a unified framework that formalizes these algorithmic contributions as operators on learned models of the environment. Our formalism reveals some synergies among these innovations, and it suggests a straightforward way to compose them. The resulting algorithm, Fitted R-MAXQ, is the first to combine the function approximation of fitted algorithms, the efficient model-based exploration of R-MAX, and the hierarchical decompostion of MAXQ.

1 Introduction

Research into *reinforcement learning* (RL) has yielded diverse techniques for more efficiently converging to rewarding behaviors in initially unknown environments, but too often these techniques are studied in isolation. Without assembling these ideas into a single algorithm, we cannot fully understand how they synergize or even conflict. In this paper, we develop a compositional framework that allows us easily to integrate three of the most important advances in RL.

The first of these advances is *model-based* RL. Early work in this direction demonstrated that summarizing an agent's experience into a model facilitates the efficient reuse of data [1]. Later work investigated how the uncertainty in the model can guide exploration, yielding the first (probabilistic) finite bounds on the amount of data required to learn near-optimal behaviors [2,3]. Still, these guarantees require that the agent exhaustively explore every state. Particularly in large domains, this exploration can be impractical.

Second, *function approximation* allows RL to cope with large or even infinite state spaces by introducing generalization. It allows an algorithm to approximate the long-term value of every action in every state using only a relatively small set of parameters. Many state-of-the-art approaches employ model-free algorithms and representations that attempt to estimate these values directly from data [4,5], but they often still rely on random exploration to acquire this data.

Third, *hierarchical decomposition* is perhaps the most intuitively appealing extension of the standard approach, since we would like to imbue our learning algorithms with the same awareness of structure that seems to allow us to cope with the extraordinary complexity of the real world. Hierarchical RL has been explored via work on temporal abstraction, in which temporally extended abstract actions allow agents to reason above the level of primitive actions [6]. However, we still do not have a complete understanding of how hierarchy benefits learning and therefore how to design or discover hierarchies.

One main contribution of this work is a formulation of important algorithms from all of these branches of RL into a concise, unified notation that makes their synergies apparent. We introduce our notation in Sect. 2 and use it to review fitted function approximation, the model-based R-MAX algorithm, and the hierarchical MAXQ framework. In Sect. 3, this powerful reformulation allows us easily to define the first algorithm for model-based, continuous-state, hierarchical RL. This algorithm, which we call Fitted R-MAXQ, constitutes our second main contribution, and we evaluate it empirically in Sect. 4. Finally, we discuss future and related work in Sect. 5 before concluding in Sect. 6.

2 Model Components

A *Markov decision process* (MDP) $\langle S, A, R, P\rangle$ comprises a finite set of states S, a finite set of actions A, a $|S||A| \times 1$ reward vector R, and a $|S||A| \times |S|$ transition matrix P. Executing action a in state s earns reward $R[sa]$ on average and transitions to state s' with probability $P[sa, s']$. We define a *policy* π as a $|S| \times |S||A|$ matrix,[1] where $\pi[s, sa]$ gives the probability of executing a in s, and where $\pi[s_1, s_2a] = 0$ for $s_1 \neq s_2$. The $|S| \times 1$ *value function* V^π maps each state s to the expected discounted reward $V^\pi[s]$ of following π from that state. The value function satisfies the Bellman equation

$$V^\pi = \pi \left(R + \gamma P V^\pi\right), \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor.

Given R and P, which comprise a *model*, a planning algorithm computes a policy π that maximizes each entry of V^π. The standard policy iteration and value iteration algorithms [7] both simply alternate between updating V^π and improving π greedily. Policy iteration updates V^π by solving (1) for a fixed π, while value iteration updates V^π by evaluating the right-hand side of (1) using the fixed previous value of V^π. Both algorithms converge to the optimal value function if $\gamma < 1$.

In the RL setting, the model is not given, but model-based algorithms estimate R and P from data. We define an *instance* $i = \langle s_i, a_i, r_i, s'_i\rangle$ as a record containing a state s_i, the action a_i executed from s_i, the one-step reward r_i earned, and the successor state s'_i. Let D be a list of instances, and define $D_a = \{i \in D \mid a_i = a\}$.

[1] The policy may thus be interpreted as a matrix transitioning states to state-action pairs.

Then the maximum-likelihood reward model $\bar{R}$ and transition model $\bar{P}$ are matrices defined as follows:

$$\bar{R}[sa] = \frac{\sum_{i \in D_a} \delta(s, s_i) r_i}{n(s, D_a)} \tag{2}$$

$$\bar{P}[sa, s'] = \frac{\sum_{i \in D_a} \delta(s, s_i)\delta(s', s_i')}{n(s, D_a)}, \tag{3}$$

where $n(s, D_a) = \sum_{i \in D_a} \delta(s, s_i)$, and δ is the Kronecker delta.

A complete RL algorithm must specify an exploration policy that guides an agent to acquire the data used to estimate the model. The R-MAX algorithm [3] maintains a set U of state-action pairs where insufficient data exists to estimate the model accurately. For state-action pairs in U, the algorithm uses an optimistic model in which the action terminates the trajectory after earning immediate reward $V^{\max}$, an upper bound on the value function. In this manner, the "unknown" state-action pairs are seen as optimal, encouraging the agent to visit them and gather data. Otherwise, the maximum-likelihood estimate of the state-action's effects is used. We can express this exploration mechanism in our matrix notation for V^π as follows:

$$V^\pi = \pi \left(UV^{\max} + (I - U)(\bar{R} + \gamma \bar{P} V^\pi) \right), \tag{4}$$

where U is represented as a $|S||A| \times |S||A|$ diagonal binary matrix where $U[sa, sa] = 1$ iff $sa \in U$. R-MAX has some appealing theoretical properties, such as a probabilistic polynomial bound on the number of times it departs from a near-optimal policy [8]. However, in practice, its thorough exploration behavior is impractical, and it only directly applies to finite MDPs where it is reasonable to gather ample data for every reachable state-action pair.

2.1 Function Approximation

Function approximation scales RL to environments where exhaustive exploration is infeasible or impossible (such as any domain with a continuous or otherwise infinite state space) by introducing the idea of generalization. A function approximator defines a family of value functions with some finite parameterization. In this paper, we focus on *averagers*, which approximate the value of a given state as a weighted average of the values of a finite subset $X \subset S$. In particular, it approximates $V^\pi[s]$ as a weighted average $\sum_{x \in X} \Phi[s, x] V^\pi[x]$, for a given $|S| \times |S|$ matrix Φ. Fitted Value Iteration [9] uses this approach by performing value iteration using the following approximation of V^π:

$$V^\pi = \pi \left(R + \gamma P \Phi V^\pi \right). \tag{5}$$

Some recent algorithms have also applied function approximation to the estimated reward and transition models. The instance-based Kernel-Based Reinforcement Learning algorithm [10] estimates a value function using the equations:

$$V^\pi = \pi \left(\hat{R} + \gamma \hat{P} V^\pi \right) \tag{6}$$

$$\hat{R}[sa] = \frac{\sum_{i \in D_a} \hat{\delta}(s, s_i) r_i}{\hat{n}(s, D_a)} \tag{7}$$

$$\hat{P}[sa, s'] = \frac{\sum_{i \in D_a} \hat{\delta}(s, s_i) \delta(s', s'_i)}{\hat{n}(s, D_a)}, \tag{8}$$

where $\hat{n}(s, D_a) = \sum_{i \in D_a} \hat{\delta}(s, s_i)$ tallies the weights given by the kernel function $\hat{\delta} : S \times S \to [0, 1]$. The kernel function determines the degree $\hat{\delta}(s, s_i)$ to which data at state s_i generalizes to the model at state s. The experiments in this paper use a Gaussian kernel, as specified in Sect. 4.1. Note that the kernel-based approximate model in (7) and (8) modify the maximum-likelihood model in (2) and (3) only by substituting weights $\hat{\delta}(s, s_i) \in [0, 1]$ in place of the exact binary indicator function $\delta(s, s_i) \in \{0, 1\}$.

2.2 Hierarchy

Given the structure we perceive in the real world, it seems natural to apply hierarchy to reinforcement learning. The MAXQ decomposition [11] and options [12] are the two most popular frameworks for hierarchical RL, which defines temporally abstract actions that represent sequences of primitive actions. MAXQ decomposes an overall learning problem using a given task hierarchy, where each abstract action is a task that induces its own individual learning problem. In contrast, the options framework formalizes an abstract action as a partial policy, which can be construed as a solution to a task. We will find it convenient to interpret an abstract action o in both ways, depending on context.

A *task* $o = \left\langle T^o, A^o, \tilde{R}^o \right\rangle$ comprises a set of terminal states $T^o \subset S$, a set of child actions or tasks A^o, and a "pseudoreward" (goal) function $\tilde{R}^o : T^o \to \mathbb{R}$. It imposes an objective onto the system defined by the state space S and the child actions A^o, which may include both other tasks and primitive actions such as those assumed by the preceding sections.[2] The task terminates upon reaching a state $s \in T^o$ and then awards itself an artificial goal value $\tilde{R}^o[s]$, where $\tilde{R}^o$ is a $|S| \times 1$ vector. We can also represent T^o as a $|S| \times |S|$ diagonal binary matrix such that $T^o[s, s] = 1$ iff o terminates upon entering s.

The optimal policy π^o for task o maximizes

$$\tilde{V}^o = T^o \tilde{R}^o + (I - T^o) \pi^o \left(R^o + \gamma P^o \tilde{V}^o \right), \tag{9}$$

where R^o and P^o are the (abstract) reward and transition matrices, respectively, for the actions in A^o. This policy π^o chooses children $c \in A^o$ in a way that maximizes a combination of one-step rewards during execution and goal values upon termination. The task value function $\tilde{V}^o$ captures this combined value, but it's also possible to compute the value function V^o that only includes the one-step rewards (and is not "contaminated" with the goal rewards). In particular, V^o is given by solving

[2] We will index the child actions A^o using c instead of a to emphasize that $c \in A^o$ may be either a task/option or a primitive action.

$$V^o = \pi^o \left(R^o + \gamma P^o (I - T^o) V^o \right) . \tag{10}$$

A key insight of MAXQ is that V^o can be interpreted as the reward model for the *option*[3] $o = \langle T^o, A^o, \pi^o \rangle$ that, when initiated in a state $s \notin T^o$, simply selects actions according to π^o until reaching a state $s' \in T^o$. Suppose that $o \in A^p$ for some parent task p. Then we can capture the insight of MAXQ as $R^p[so] = V^o[s]$. In other words, we can use V^o to construct part of the reward vector for any MDP learning task p that includes o as an executable action.

The same recursive approach can also apply to the transition function. Just as the abstract reward function for an option specifies the expected (discounted) sum of one-step rewards earned before reaching a terminal state, the abstract transition function for an option should specify the expected (discounted) probability of terminating in each terminal state. To this end, we define the $|S| \times |S|$ *terminal-state matrix* Ω^o for an option o with the following Bellman-like equation:

$$\Omega^o = \pi^o \left(P^o T^o + \gamma P^o (I - T^o) \Omega^o \right) . \tag{11}$$

Note intuitively that each column of Ω^o can be interpreted as a value function for a task which gives a reward of 1 upon terminating in the state corresponding to that column. As a result, Ω^o can be computed using standard MDP planning algorithms. Finally, we observe that if $o \in A^p$ for some parent task p, then $P^p[so, s'] = \Omega^o[s, s']$. Here, P^p is a multi-time model [12], so its rows may not sum to 1, reflecting the effect of the discount factor over time. This representation thus folds the duration of actions (typically represented explicitly in the standard SMDP formalism) into the discounted transition probabilities.

Our hierarchical decomposition thus specifies how to construct the reward and transition matrices for a task p recursively given the value functions and terminal-state matrices of the options $o \in A^p$. To complete this recursive specification, we need only give the base case. For a primitive action a, the value function V^a and terminal-state matrix Ω^a correspond exactly to the reward and transition models for that action.

3 Compositional Algorithms

The algorithms described in Sect. 2 generate exploration policies by solving modified forms of the standard Bellman equation (1). However, each of the modified equations (4), (5), and (9) share the same general form of (1): we can construe the right-hand side as π multiplied by the sum of a $|S| \times 1$ vector (that doesn't depend on V^π) and a $|S| \times |S|$ matrix multiplied by V^π. These equations are therefore equivalent to the standard Bellman equations for a modified version of the original Markov decision process.[4]

[3] It is straightforward to support the stochasic termination of the standard options framework by defining the diagonal entries of T as the termination probabilities.

[4] Equation (9) can be rewritten into this form by handling termination after applying the policy, but the representation of T^o and $\tilde{R}^o$ is then less intuitive.

We formalize such modifications as follows. For a given set of states S and primitive actions A, let $\mathcal{D}$ be the space of all possible lists of instances, $\mathcal{R}$ be the space of all possible $|S| \times 1$ reward vectors, and $\mathcal{P}$ be the space of all possible $|S| \times |S|$ transition matrices. The $|S||A| \times 1$ reward vector for a given learning task o is then obtained by composing the reward vectors for each child action $c \in A^o$ in the appropriate way. Similarly, the $|S||A| \times |S|$ task transition matrix is composed from the $|S| \times |S|$ action transition matrices. We now define a *model generator* $G : \mathcal{D} \to \mathcal{R} \times \mathcal{P}$ as a mapping from a list of instances to a reward and transition model for a given primitive action, and a *model operator* $M : \mathcal{R} \times \mathcal{P} \to \mathcal{R} \times \mathcal{P}$ as a mapping from one reward and transition model to another.

We can formalize the maximum-likelihood model as a family of model generators $\textsc{mle}_a$ for each primitive action a:

$$\textsc{mle}_a(D) = \left(\bar{V}^a, \bar{\Omega}^a\right), \tag{12}$$

where $\bar{V}^a[s]$ and $\bar{\Omega}^a[s, s']$ are given by the right-hand sides of (2) and (3), respectively. The R-MAX algorithm then generates an exploration policy by using a standard planning algorithm on the learning task composed with the action models $\textsc{r-max}_a(\text{MLE}_a(D))$, where

$$\textsc{r-max}_a(R, P) = (U_a V^{\max} + (I - U_a)R, (I - U_a)P), \tag{13}$$

where U_a is the $|S| \times |S|$ submatrix of U such that $U_a[s, s] = 1$ iff sa is unknown. Note that the R-MAX operator is defined as a function of the data D, which are required to define U.

The Fitted R-MAX algorithm, which extends R-MAX using the model approximation of KBRL and fitted planning [13], can now be seen as planning with the action models $\textsc{fvi}(\textsc{r-max}_a(\textsc{kbrl}_a(D)))$, where

$$\textsc{fvi}(R, P) = (R, P\Phi) \tag{14}$$

encapsulates Fitted Value Iteration for a given Φ, and

$$\textsc{kbrl}_a(D) = \left(\hat{V}^a, \hat{\Omega}^a\right), \tag{15}$$

where $\hat{V}^a[s]$ and $\hat{\Omega}^a[s, s']$ are given by the right-hand sides of (7) and (8), respectively.

R-MAXQ, another recent extension to R-MAX, incorporates the MAXQ-based model decomposition described in Sect. 2.2 [14]. For a given task o, the computation of the option policy π^o requires planning with modified action models $\textsc{maxq}^o(V^c, \Omega^c)$ for each child $c \in A^o$, where

$$\textsc{maxq}^o(R, P) = \left(T^o \tilde{R}^o + (I - T^o)R, (I - T^o)P\right). \tag{16}$$

Given the option policy π^o, as well as policies for each descendent of o, the MAXQ decomposition also defines a model (V^o, Ω^o) of o. In this sense, each task o defines a model generator (since each option policy is a function of D).

Algorithm 1. MODEL(c)

```
if $c$ is a (primitive) action then
    $a \leftarrow c$
    $(V^a, \Omega^a) \leftarrow G^a(D)$
    Return $(V^a, \Omega^a)$
else {$c$ is a (composite) task/option}
    $o \leftarrow c$
    $(\pi^o, R^o, P^o) \leftarrow$ PLAN($o$)
    $V^o \leftarrow$ solution to $V^o = \pi^o(R^o + \gamma P^o(I - T^o)V^o)$
    $\Omega^o \leftarrow$ solution to $\Omega^o = \pi^o(P^o T^o + \gamma P^o(I - T^o)\Omega^o)$
    Return $(V^o, \Omega^o)$
end if
```

Algorithm 1 precisely defines this model generator. Note that Algorithms 1 and 2 are mutually recursive, and they assume the following global parameters: a set of instances D, model generators G^a for each primitive action a, and model operators M^o for each task o. Algorithm 2 constrains the planning algorithm to only execute an option in states not in the option's termination set. In other words, for all options o and parent tasks p, $pi^p[s, so] > 0$ implies $s \notin T^o$. (The option's initiation set is the complement of its set of terminal states.)

Algorithm 2. PLAN(o)

```
for all children $c \in A^o$ do
    $(V^c, \Omega^c) \leftarrow M^o($MODEL$(c))$
end for
$R^o \leftarrow$ choose so that $R^o[sc] = V^c[s]$
$P^o \leftarrow$ choose so that $P^o[sc, s'] = \Omega^c[s, s']$
$\pi^o \leftarrow$ optimize $\tilde{V}^o = \pi^o \left(R^o + \gamma P^o \tilde{V}^o\right)$        {subject to initiation constraints}
Return $(\pi^o, R^o, P^o)$
```

Given these subroutines, Algorithm 3 describes a broad family of model-based RL algorithms parameterized by a task hierarchy, model generators attached to each primitive action, and model operators attached to each task. For example, suppose that for a given task hierarchy we define $G^a = \text{R-MAX}_a \circ \text{MLE}_a$ for each primitive action a and $M^o = \text{MAXQ}$, for each task o. Then running EXECUTE on the root task of this hierarchy is exactly equivalent to R-MAXQ. Note that this algorithm uses the same model (R^o, P^o) to compute both the "contaminated" value function $\tilde{V}^o$ and the real value function V^o. It can use this model for both purposes, since the additional goal-reward values only affect the reward vector at terminal states. Equation (10) disregards values of terminal states, and Algorithm 2 prevents the execution of the option at terminal states.

These procedures can also implement non-hierarchical algorithms by using a "flat" hierarchy. Let A be the set of available primitive actions and define the task `Root` such that $A^{\texttt{Root}} = A$ and $T^{\texttt{Root}} = 0$. If we define $G^a = \text{R-MAX}_a \circ \text{MLE}_a$

and $M^{\texttt{Root}} = \textsc{i}$, the identity operator, then $\textsc{Execute}(\texttt{Root})$ is exactly equivalent to the original R-MAX algorithm. If we instead define $G^a = \textsc{r-max}_a \circ \textsc{kbrl}_a$ and $M^{\texttt{Root}} = \textsc{fvi}$, then we obtain the Fitted R-MAX algorithm.

Algorithm 3. $\textsc{Execute}(c)$

```
if $c$ is a (primitive) action then
  $a \leftarrow c$
  Execute action $a$ in the environment
  $r \leftarrow$ reward
  $s' \leftarrow$ successor state
  $D \leftarrow D \cup \{\langle s, a, r, s' \rangle\}$
else {$c$ is a (composite) task/option}
  $o \leftarrow c$
  repeat
    $s \leftarrow$ current state
    $(\pi^o, V^o, \Omega^o) \leftarrow \textsc{Plan}(o)$          {$V^o$ and $\Omega^o$ ignored}
    $c \leftarrow$ choose child action/option according to $\pi^o$
    $\textsc{Execute}(c)$
  until $T^c[s', s'] = 1$
end if
```

3.1 Fitted R-MAXQ

Our compositional approach to model-based RL immediately suggests a novel algorithm, obtained by applying all of our available model operators. We define $G^a = \textsc{r-max}_a \circ \textsc{kbrl}_a$, to obtain the optimistic exploration of R-MAX and the instance-based generalization of primitive action models of Kernel-Based Reinforcement Learning. We define $M^o = \textsc{maxq} \circ \textsc{fvi}$ to obtain the subtask decomposition of MAXQ and the value function approximation of Fitted Value Iteration. In keeping with prior algorithms extending R-MAX, we refer to this novel algorithm as Fitted R-MAXQ. For concreteness, we specify Fitted R-MAXQ in Algorithm 4, which also optimizes the computation of Algorithms 1–3 by using dynamic programming to unroll the mutual recursion. Note that Algorithm 4 assumes a continuing task, but the modifications for episodic tasks are straightforward.[5]

To our knowledge, Fitted R-MAXQ is the first model-based RL algorithm to combine function approximation and hierarchical decomposition. Interestingly, a close inspection reveals notable structural similarities among the operators that comprise Fitted R-MAXQ, which creates opportunities for synergies. For example, consider the R-MAX and MAXQ operators, (13) and (16). Both modify a model by changing the rewards at a subset of the states, which also become terminal. In the case of R-MAX, this subset is the set of unknown states U_a; for MAXQ, it is the set of terminal states T^o. Entering this set terminates the

[5] The $T^{\texttt{Root}}$ matrix should remain zero even in episodic tasks, since it is the environment and not the agent terminating.

trajectory (by assigning zero probability to every successor) but earns a final reward given by the upper bound $V^{\max}$ or the pseudoreward function $\tilde{R}^o$. The net effect of these operators is to bias the exploration policy of the algorithm, towards leaving the known set and towards manually specified subgoal states, respectively. This characterization leads us to conjecture that one important role of hierarchy is to focus the otherwise too thorough exploration of R-MAX by requiring the optimistic value of exploration to overcome the perceived subgoal rewards.

Algorithm 4. FITTED-R-MAXQ(`Root`)

```
Initialize $stack$ to [Root]
loop
  $s \leftarrow$ current state
  $c \leftarrow$ top of $stack$
  while $c$ is a task, not a primitive action do
    $o \leftarrow c$
    $c \leftarrow$ choose child in $A^o$ using $\pi^o$
    Push $c$ onto $stack$
  end while
  $a \leftarrow c$
  Execute action $a$
  $r \leftarrow$ one-step reward
  $s' \leftarrow$ successor state
  $D \leftarrow D \cup \{\langle s, a, r, s' \rangle\}$
  $V^a \leftarrow U_a V^{\max} + (I - U_a)\hat{V}^a$          {$\hat{V}^a[s]$ equals right-hand side of (7)}
  $\Omega^a \leftarrow (I - U_a)\hat{\Omega}^a$              {$\hat{\Omega}^a[s, s']$ equals right-hand side of (8)}
  for all tasks $o$ do {bottom-up}
    $R^o \leftarrow$ construct so that $R^o[sc] = V^c[s]$
    $P^o \leftarrow$ construct so that $P^o[sc, s'] = \Omega^c[s, s']$
    $\pi^o \leftarrow$ optimize $\tilde{V}^o = T^o \tilde{R}^o + (I - T^o)\pi^o(R^o + \gamma P^o \Phi V^o)$
    $V^o \leftarrow$ solve $V^o = \pi^o(R^o + \gamma P^o (I - T^o) V^o)$
    $\Omega^o \leftarrow$ solve $\Omega^o = \pi^o(P^o T^o + \gamma P^o (I - T^o)\Omega^o)$
  end for
  repeat {$stack$ begins with a primitive on top}
    Pop $stack$
    $o \leftarrow$ top of $stack$
  until $T^o[s', s'] = 0$                                   {$s'$ not terminal}
end loop
```

We have already seen that some model operators modify the transition matrix by forcing some states to be terminal, but the FVI operator (14) instead "redirects" transitions into a relatively small subset $X \subset S$. By construction, at most $|X|$ of the columns of Φ are nonzero. Since this matrix is multiplied to the right of the original transition matrix, at most $|X|$ of the rows of V^π affect the Bellman equation. Even when S is infinite, this property permits our implementation to employ a sparse representation of the matrices involved. In general, in any one time step, the algorithm must store only $|X| + 1$ rows of each matrix, one for the current state and the rest for X.

Interestingly, the MAXQ decomposition can have a similar effect on planning efficiency. Note that (11) defines the abstract transition matrix for an option o in such a way that the nonzero columns correspond to the terminal states in T^o. Since many tasks achieve subgoal states that comprise a small fraction of the state space, planning at the abstract level of tasks may permit a compact representation of the value function which compounds with any reductions due to fitted function approximation.

4 Experiments

In order to exercise the full capabilities of Fitted R-MAXQ, we introduce a new domain modeled after the RL benchmark environment Puddle World. Puddle World already has a continuous state space, and no modifications are necessary to enable model-based reasoning (which is purely an algorithmic issue), but we introduce a task hierarchy that enables hierarchical reasoning. We intend our extensions to give the overall task more structure of the sort found in real-world tasks.

First, we describe the original Puddle World environment, depicted in Fig. 1. The agent must navigate the unit square to reach a goal state in the upper-right corner, which terminates each episode. Four primitive actions move the agent 0.05 in each of the four cardinal directions, with some Gaussian noise ($\sigma = 0.01$) added to each of the two state variables after every action. Each action incurs a -1 penalty until reaching the goal, but each time step spent in a puddle incurs an additional penalty between 0 and -40, depending on the proximity to the middle of the puddle.

We modify this environment by removing the goal state in the corner and instead giving the agent a set of four different resources it must harvest in each

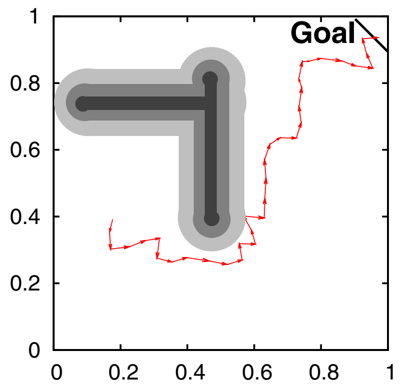

Fig. 1. A trajectory in the original Puddle World environment

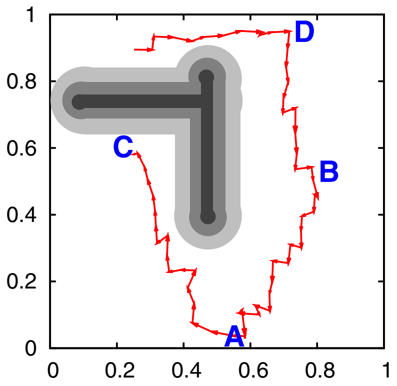

Fig. 2. A trajectory in the modified Puddle World environment. The agent gathers resources D, B, A, then C, but any order is permissible.

episode, as shown in Fig. 2. Each resource can only be collected in the neighborhood (within distance 0.1) of a specific spot in the unit square, which is initially unknown. For each resource, a binary state variable tracks whether the resource has been collected, and a distinct primitive action allows the agent to search the current location for the resource and harvest it if present. In each episode, the agent begins in a random location, so over time it must learn the locations of the resources, the locations of the costly puddles, and how to harvest all four resources as cheaply as possible. This environment has six state variables, two continuous and four binary, and eight primitive actions, four movement actions and the four collection actions.

We ran each algorithm tested for 50 independent trials, using for each algorithm the same set of 50 configurations of resource locations, generated uniformly at random but with no location inside of a puddle. For each configuration, we generated a fixed sequence of 500 start states, again uniformly at random. Each trial lasted for 500 episodes, and we limited each episode to 1000 time steps.

4.1 Algorithm Configurations

We compare several different instantiations of our compositional framework for model-based RL. In this section, we describe the precise configuration of each model generator and model operator used. We use value iteration with prioritized sweeping [1] both to compute the optimal policy π^o given a model (R^o, P^o) and also to evaluate π^o to obtain the abstract model (V^o, Ω^o). We used a discount factor of $\gamma = 1$, since the task is episodic.

The maximum-likelihood model generator MLE has no parameters, but to apply finite algorithms in PuddleWorld we used a discretization of the unit square into a 16×16 grid. This discretization seems quite coarse, but finer grids only lead to more excessive exploration without a concomitant increase in policy quality.

For the instance-based model approximation of KBRL, we adopted a Gaussian kernel function:

$$\hat{\delta}(s_1, s_2) = e^{-(d(s_1,s_2)/b)^2}, \tag{17}$$

where $d(s_1, s_2)$ is the Euclidean distance between s_1 and s_2, and $b = \frac{1}{16}$ is a bandwidth parameter that controls the breadth of generalization, chosen using coarse optimization. To compute $\hat{\delta}$ efficiently, we stored the instances in a cover tree [15] and rounded down to zero any value of $\hat{\delta}(s_1, s_2) < 0.01$. Finally, we adopt the "relative transition model" of [13], which modifies (8) by using the vector displacement observed at instance i instead of the absolute successor state observed:

$$\hat{P}[sa, s'] = \frac{\sum_{i \in D_a} \hat{\delta}(s, s_i)\delta(s', s + (s'_i - s_i))}{\hat{n}(s, D_a)}. \tag{18}$$

All of the algorithms we tested rely on the R-MAX approach to exploration. We set $V^{\max} = 0$, since all the immediate rewards in Puddle World are negative. When used with MLE, we defined $U_a = \{s \in S \mid n(s, D_a) < 2\}$. Since the stochasticity in Puddle World is relatively benign, gathering more data for each state-action didn't improve the final policy quality but resulted in much more expensive exploration. When used with KBRL, we defined $U_a = \{s \in S \mid \hat{n}(s, D_a) < 1\}$. This low threshold seemed adequate since KBRL must typically generalize from several instances to reach a kernel weight of 1.

For FVI, we defined the averager Φ using linear interpolation over a uniformly spaced grid, with a resolution of $\frac{1}{16}$. This function approximation scheme therefore approximates the value of a point in the unit square (for a particular setting of the binary state variables) as an interpolation between the four surrounding points. Again, increasing the resolution did not improve the quality of the learned policy, but it did increase the computational burden of planning.

For the hierarchical algorithms, we defined a simple task hierarchy for our modified Puddle World that corresponds to the prior knowledge that the four resource collection actions are independent of one another.[6] For each resource, we define a task o such that the children actions A^o include the four movement primitives and the action that collects that resource. The terminal set T^o includes all states where the resource's boolean flag is set, and $\tilde{R}^o = 0$. The root of the hierarchy has these four tasks as children; it cannot execute any primitive actions directly.

Finally, all the algorithms benefitted from state abstraction. The four primitive movement actions neither depend on nor affect the boolean state variables for the four activities. Similarly, each activity only depends on the coordinates and only affects the corresponding boolean state variable. Due to limitations on space, we omit the details of these state abstractions, which were implemented in the obvious way.

[6] This hierarchy therefore imparts less domain knowledge than the hierarchy Dietterich provided for learning in the Taxi domain [11], where the possible passenger coordinates were all known a priori.

4.2 Results

Figure 3 shows learning curves for four algorithms: R-MAX, R-MAXQ, Fitted R-MAX, and our combination of these algorithms, Fitted R-MAXQ. All four algorithms converge to statistically the same policy quality after only 25-30 episodes, but they incur very different exploration costs before getting there.

Figure 4 integrates under the curves in Fig. 3 to show the total learning costs. Note that both figures only show the first several episodes, to focus on the period of learning when the algorithms' performance differs. Note that the benefit of adding both hierarchical decomposition and function approximation to R-MAX is greater than the sum of the benefits for adding each innovation by itself!

An inspection of the behavior of Fitted R-MAXQ reveals that it outperforms the other algorithms largely by avoiding excessive exploration in the puddles. Consider a state in the middle of a puddle that is in the set of unknown states U_a for some primitive action a. The R-MAX operator will assign this state the optimistic value $V^{\max}$, but this value does not guarantee that the agent will attempt to reach this state. If the predicted cost of completing the current task is smaller than the predicted cost of wading through the puddle to the unknown state, the agent will choose to ignore the unknown state and instead exploit a path through known states. In the non-hierarchical case, the current task is always to complete all the remaining activities, which may have a rather high cost. In the hierarchical case, the current task is to complete a particular one

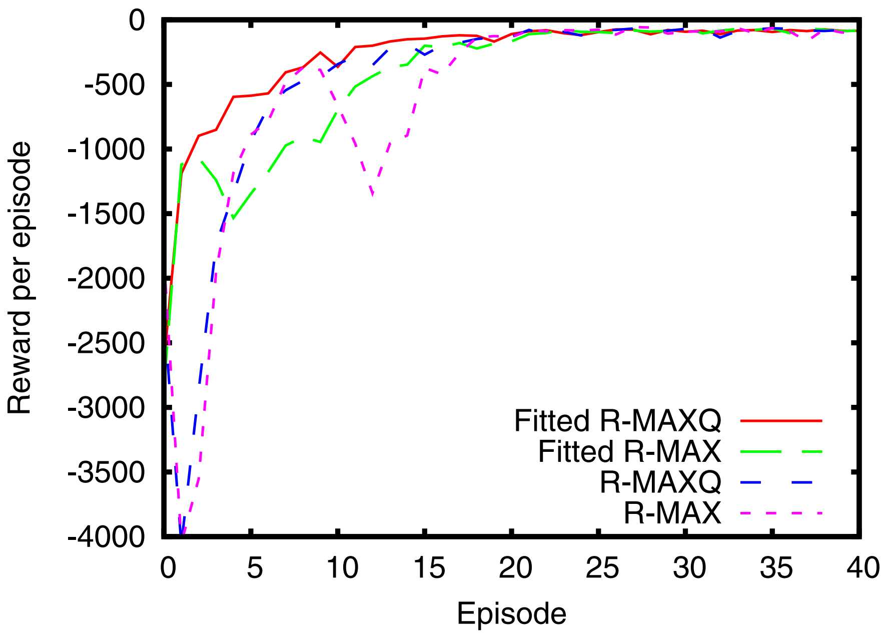

Fig. 3. Reward per episode for variations of R-MAX with and without function approximation and hierarchical decomposition

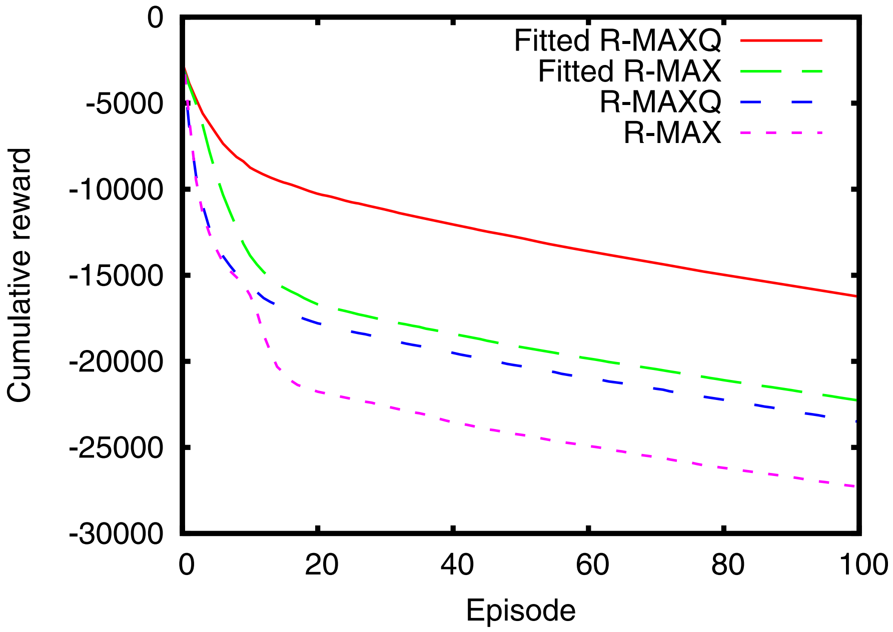

Fig. 4. Cumulative reward for variations of R-MAX with and without function approximation and hierarchical decomposition

of the activities, which is more likely to have a lower cost than wading into the puddle. In a sense, the hierarchical decomposition limits the optimism applied to unknown states, which R-MAXQ models as terminating only the current task, not the entire episode. Meanwhile, discretization interferes with the accurate prediction of the costs of exploring versus exploiting. The coarse discretization we used is very effective in most of the state space, where the dynamics are the same, but not near the puddles, where the immediate reward varies quickly as a function of the coordinates.

5 Discussion

In this paper, we cast certain existing algorithms into a unified framework with an eye towards defining a new algorithm that combined all the desired existing features. The generality of our framework leaves open the possibility that still more algorithms can be formalized in terms of model generators and model operators. Investigating combinations of these algorithms can only help us to develop deeper understandings of their individual contributions in context.

One important direction for future work is to derive general properties of the class of algorithms defined by our compositional framework. Each algorithm in this class behaves strictly according to the optimal policy for an MDP derived in some way from data. An understanding of the expressivity and limitations of such algorithms might either inspire the creation of new operators within this framework or of new algorithms that meaningfully break out of it. For example, hierarchical RL requires the ability to work with a derived MDP that has a different action space than the original MDP. We observe that the problem of optimal exploration in RL can be reduced to a planning problem in a derived MDP, where the state space is augmented with beliefs concerning the underlying MDP [16]. In what ways can model operations productively change the state and action spaces of the underlying MDP? Finally, integration with the ongoing work on MDP homomorphisms [17] may allow our framework to deal more explicitly with state abstraction.

6 Conclusion

This paper developed two main contributions to the literature on scaling reinforcement learning to increasingly complex environments. First, it introduced a novel, unifying notation and formulation of three previously disjoint ideas in RL: model-based exploration, function approximation, and hierarchy. This formulation construed existing algorithms as essentially the application of a standard planning algorithm to a transformed reward and transition model. Second, the paper leveraged this new notation to unify these three ideas into Fitted R-MAXQ, the first algorithm for hierarchical, model-based RL in continuous domains. Fitted R-MAXQ is fully implemented and evaluated in a hierarchical Puddle World, significantly outperforming algorithms that utilize only a subset of its components.

References

1. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning 13, 103–130 (1993)
2. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 260–268 (1998)
3. Brafman, R.I., Tennenholtz, M.: R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3, 213–231 (2002)
4. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research 4, 1107–1149 (2003)
5. Riedmiller, M.: Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In: Proceedings of the European Conference on Machine Learning (2005)
6. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete-Event Systems 13, 41–77 (2003); Special Issue on Reinforcement Learning
7. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., Chichester (1994)
8. Kakade, S.M.: On the Sample Complexity of Reinforcement Learning. PhD thesis, University College London (2003)

9. Gordon, G.J.: Stable function approximation in dynamic programming. In: Proceedings of the Twelfth International Conference on Machine Learning (1995)
10. Ormoneit, D., Sen, Ś.: Kernel-based reinforcement learning. Machine Learning 49(2), 161–178 (2002)
11. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research 13, 227–303 (2000)
12. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112(1–2), 181–211 (1999)
13. Jong, N.K., Stone, P.: Model-based exploration in continuous state spaces. In: Proceedings of the Seventh Symposium on Abstraction, Reformulation and Approximation (2007)
14. Jong, N.K., Stone, P.: Hierarchical model-based reinforcement learning: R-MAX + MAXQ. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (2008)
15. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: Proceedings of the Twenty-Third International Conference on Machine Learning (2006)
16. Duff, M.: Design for an optimal probe. In: Proceedings of the Twentieth International Conference on Machine Learning, pp. 131–138 (2003)
17. Ravindran, B., Barto, A.G.: SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (2003)