Composable Modular Reinforcement Learning

Christopher Simpkins, Charles Isbell
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280, USA
chris.simpkins@gatech.edu, isbell@cc.gatech.edu

Abstract

Modular reinforcement learning (MRL) decomposes a monolithic multiple-goal problem into modules that solve a portion of the original problem. The modules' action preferences are arbitrated to determine the action taken by the agent. Truly modular reinforcement learning would support not only decomposition into modules, but composability of separately written modules in new modular reinforcement learning agents. However, the performance of MRL agents that arbitrate module preferences using additive reward schemes degrades when the modules have incomparable reward scales. This performance degradation means that separately written modules cannot be composed in new modular reinforcement learning agents as-is – they may need to be modified to align their reward scales. We solve this problem with a Q-learning-based command arbitration algorithm and demonstrate that it does not exhibit the same performance degradation as existing approaches to MRL, thereby supporting composability.

Introduction

Decomposition is an important tool in designing software systems in general and, as we will see in the next section, an important tool for dealing with the larger state spaces likely to be encountered in real-world problems. Hierarchical reinforcement learning (HRL), discussed in the Related Work section, decomposes reinforcement learning problems temporally, modeling intermediate tasks as higher-level actions. HRL has also formed the basis of reinforcement learning-based programming systems.

MRL decomposes the original problem concurrently, modeling an agent as a set of concurrently running reinforcement learning modules. MRL has been used primarily to model multiple-goal problems and to deal with large state spaces. We would like to use MRL as a basis for reinforcement learning-based programming systems but, as we discuss in the section on composability in modular reinforcement learning, current approaches to MRL have a serious limitation – a requirement for module reward scale comparability – which limits their usefulness in programming systems. The work we present in this paper solves the reward scale comparability problem.

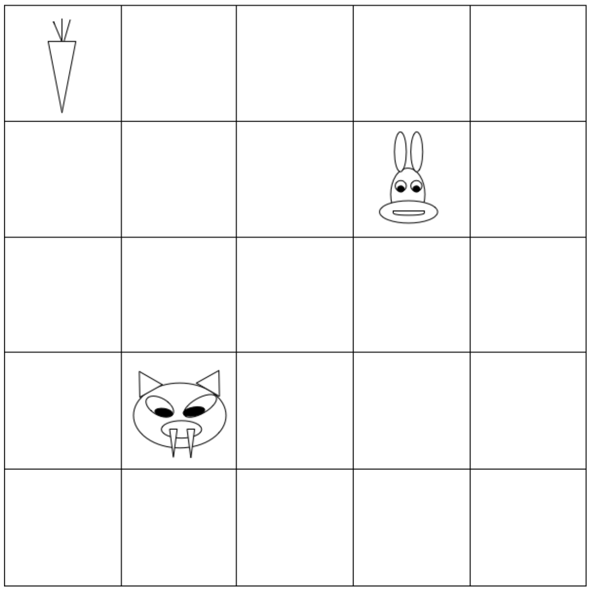

Figure 1: In the grid world above, the bunny must pursue two goals simultaneously: find food and avoid the wolf. The bunny may move north, south, east, or west. When it finds food it consumes the food and new food appears elsewhere in the grid world, when it meets the wolf it is eaten and "dies."

The Curse of Dimensionality

As with other kinds of machine learning, reinforcement learning must deal with the curse of dimensionality. In reinforcement learning the curse of dimensionality manifests itself primarily in the size of the state space, namely, the state space grows exponentially in the number of state features.

As an example, consider the 5×5 grid of the bunny world in Figure 1. The bunny, food, and wolf can be in one of 25 possible locations. If the task of the bunny were only to reach a particular location, then the number of states would be 25. Add the task of avoiding a wolf and the state space grows to $25^2 = 625$. Add finding food and the state space grows to $25^3 = 15625$. If you model this problem as two separate modules, one in which the bunny avoids the wolf and another in which the bunny finds food, then each module solves a problem with a state space of $25^2 = 625$ states (bunny plus wolf and bunny plus food).

Modular Reinforcement Learning

An MRL (Russell and Zimdars 2003; Sprague and Ballard 2003b) agent is decomposed into several modules each of which concurrently learns a different subgoal of the original, complex, multiple-goal learning problem. In an MRL agent each module observes the action taken by the agent, the state transition and a reward signal specific to the module. At each time step, the agent combines the action preferences of the modules to compute the joint policy. In most MRL algorithms the joint policy is derived from a joint Q-function in which the Q-values for each module are added:

$$Q_{joint}(s, a) = \sum Q_i(s_i, a) \tag{1}$$

where each module m_i has its own state abstraction, s_i for each $s \in S$.

Russell's and Zimdars' Q-decomposition (Russell and Zimdars 2003) is equivalent to Sprague and Ballard's GM-Sarsa (Sprague and Ballard 2003b). Both of these investigations of Q-function decomposition showed that Sarsa is a better choice for agent modules than Q-learning because Sarsa is on-policy. Sarsa updates its Q-function based on the policy being followed by the agent while Q-learning updates its Q-function assuming the module's locally optimal policy will be followed. Because modules do not have direct control over the policy being followed, modules using on-policy learning algorithms like Sarsa perform better because the global action is communicated to the modules to update local Q-functions. Because we evaluate our algorithm using the same problem as Sprague and Ballard, in this work we will refer mostly to GM-Sarsa.

There is clearly a desire to model agents with multiple goals represented as reinforcement learning modules. We list only a few examples here. Sprague and Ballard have applied GM-Sarsa to problems in eye movement scheduling (Sprague and Ballard 2003a; Sprague, Ballard, and Robinson 2007). Konidaris and Barto applied an algorithm derived from GM-Sarsa to adaptive robot control (Konidaris and Barto 2006). Aissani and colleagues used GM-Sarsa to develop a system for dynamic scheduling of maintenance tasks in the petroleum industry (Aissani, Beldjilali, and Trentesaux 2009; Chaari et al. 2014). Rowe and colleagues used GM-Sarsa for interactive narrative planning (Rowe and Lester 2013).

All of the work applying MRL has been done by single teams of researchers applying MRL to research problems. Thus, modules were authored together with comparable reward scales. To support reusability in a software engineering sense we need to support the separate authoring of modules. Separately authored modules may use reward scales that are internally consistent within modules but incomparable to the rewards used in other modules. In this paper we show that existing approaches to MRL degrade when modules use incomparable reward scales and present an algorithm that does not exhibit the same degradation.

Composability in Modular Reinforcement Learning

Our ultimate goal with modular reinforcement learning is to facilitate the integration of reinforcement learning into a programming language in order to support intelligent agent software engineering, as we outlined in earlier work (Simpkins et al. 2008). To support software engineering we would like MRL components to be truly modular. In particular, we would like the components to be reusable without modification – composable. Unfortunately, current approaches implicitly require a global, monolithic reward signal, which detracts from composability. In particular, an agent programmer cannot locally define the reward function for a module because the reward scales of other modules must be taken into account. For example, if a Bunny (Figure 1) agent programmer creates a FindFood module and then decides to use an AvoidWolf module written by another programmer but the AvoidWolf module uses a reward scale several times higher than FindFood, then a MRL algorithm that uses additive rewards will favor AvoidWolf to the near exclusion of FindFood. One of the modules would need to be modified to align the reward scales of all the modules. As we summarize briefly in the following section, Bhat and colleagues showed that this limitation is inherent in existing approaches to modular reinforcement learning (Bhat, Isbell, and Mateas 2006).

Ideal Action Selection is Impossible

Aside from the practical challenges cited above, Bhat, et., al. (Bhat, Isbell, and Mateas 2006) showed that ideal action selection from modules that vote on the agent's single action is impossible in full generality because the problem reduces to Arrow's Paradox (Arrow 1963): an agent is a "society" of modules, and action selection is social choice. The problem is that we want the action selection mechanism to have the following reasonable properties:

- **Universality**: the ability to handle any possible set of modules.
- **Unanimity**: guarantee that if every module prefers action A, action A will be selected.
- **Independence of Irrelevant Alternatives**: each module's preference for actions A and B are independent of the availability of any other action C. This property prevents any particular module from affecting the global action choice by dishonestly reporting its own preference ordering.
- **Scale Invariance**: ability to scale any module's Q-values without affecting the arbitrator's choice. This is the crucial property that allows separately authored modules with incomparable reward signals.
- **Non-Dictatorship**: no module gets its way all the time.

If $|A| \geq 3$, then there does not exist an arbitration function that satisfies each of the properties listed above (Roberts 1980). So even simple agents with more than three actions are too complex for theoretically ideal arbitration. This paper contributes a novel formulation of MRL and an algorithm that implements it.

Reformulating MRL

Bhat, et., al., (Bhat, Isbell, and Mateas 2006) argued for a "benevolent dictator", a special module executing a command arbitration function for action selection but left the arbitration algorithm unspecified. Here we present a command arbitration algorithm embodying the ideas in (Bhat, Isbell, and Mateas 2006) and show that it performs competitively with other MRL algorithms and shows superior robustness to module modification. This robustness to module modification is the chief enabler of truly modular reinforcement learning in which modules can be transferred from one system to another without having to re-engineer the reward signals to fit the new host system.

This formulation relaxes the non-dictatorship requirement of ideal action selection if you think of the arbitrator as a special module. By Arrow's theorem, other properties will still hold. In the next section we present a reformulation of MRL based on this idea.

Formalization

Our reformulation of MRL adds a *command arbitrator* which has a state space that may be the same as or different from the modules' state spaces, an action set that represents choosing a module, $A_{CA} = 1...n$ for an agent with n modules, and a reward signal that represents the agent's overall goal. The arbitrator's reward function, $R_{CA}(s)$, is independently defined rather than being derived from the module rewards. Note that $R_{CA}(s)$ may or may not be equal to the sum of the rewards of the agent's modules. In fact, the modules' rewards, $R_i(s, a)$, may not have any relation to the arbitrator's reward $R_{CA}(s)$.

The agent's policy is defined indirectly by the arbitrator's policy, $\pi_{CA}(s, a)$, which assigns probabilities to the selection of a module for each state, where the selected module's preferred action becomes the agent's action in that state.

For the agent author, this formulation adds the requirement of authoring a dedicated reward signal for the arbitrator. For our bunny agent, this is LiveLongProsper:

- *Why* avoid predator, *why* eat? To live longer.
- Encodes the tradeoffs between modules – perhaps food is more important to some bunnies.
- The arbitrator could be hand-authored, or could be another RL agent.

For the small cost of authoring a reward signal that represents the "greater good" you get true modularity, that is, the ability to combine separately authored modules with incomparable rewards. This new reward signal is now the metric we use to measure the performance of the agent.

In our MRL framework an agent is an arbitrator plus a list of modules. Formally, an agent consists of the following elements (we use CA in subscript to refer to the command arbitrator and numbers or i to refer to modules):

1. A reward function for the command arbitrator, $R_{CA}(s)$,
2. An action set A for the agent as a whole, shared by each module,
3. A set of reinforcement learning modules, M
4. A state abstraction function, $moduleState_i$ for each module m_i
5. A reward function, $R_i(s)$ for each module m_i

In the next section we present our Arbi-Q algorithm which implements the arbitrator in this framework.

The Arbi-Q Command Arbitration Algorithm

Our command arbitrator is itself a reinforcement learner. The state space for the arbitrator may be the world state, in which no state abstraction is used for the arbitrator, or may employ a state abstraction like the modules. The arbitrator's action set, A_{CA}, is a set of integer indexes to the agent's list of modules. As with any reinforcement learner, the arbitrator learns a policy. In the case of the arbitrator this policy, π_{CA} is a mapping from states to modules. The modules' policies are mappings from abstracted module state to actions in the world, that is, the agent's actions. The arbitrator's policy defines which module chooses the agent's action in a particular state.

Our Arbi-Q implementation uses the Sarsa algorithm (Rummery and Niranjan 1994) to learn the arbitrator's policy, but any reinforcement learning algorithm may be used. At each time step the arbitrator uses its policy to select a module, then the module uses its local policy to select an action that the agent executes. The results of executing the action are communicated to the arbitrator as a consequence of the module selection, and to the modules as a consequence of action selection. Each module uses a state abstraction function to transform the world state into the the subset of the state relevant to the module, and a reward function that is based on the module's state abstraction. In this way the modules are coupled to the world in which they operate – the modules can only operate in worlds which contain the state features expected by its state abstraction function – but the modules are not coupled to other modules or to an arbitrator. The Arbi-Q algorithm is sketched in Algorithm 1. While the algorithm presented here shows the modules learning at the same time the arbitrator is learning, it is also possible to pre-train the modules and hold them constant while training the arbitrator. Because Arbi-Q uses Sarsa to learn its module-selection policy it inherits the convergence properties of Sarsa (Singh et al. 2000).

Algorithm 1 Arbi-Q (sketch)

```
$Q_{CA} \leftarrow$ random initial values
for each module $i$ do
    $Q_i \leftarrow$ random initial values
end for
for each time step do
    Command arbitrator chooses a module $m$ from $Q_{CA}$
    $a \leftarrow \epsilon-$greedy action for $s_m$ from $Q_m$
    Execute $a$, observe effects $r_{CA}$ and $s'$
    Update $Q_{CA}(s_{CA}, m)$ from action $m$ and $R_{CA}(s_{CA})$
    for each module $i$ do
        Update $Q_i(s_i, a)$ from action $a$ and $R_i(s_i, a)$
    end for
end for
```

Experiments

Our principal claim is that Arbi-Q is robust to modules with incomparable reward scales, which would be an authoring error in existing MRL approaches. Our experiments show that GM-Sarsa/Q-decomposition degrades when modules are modified to have incomparable reward scales and that Arbi-Q is robust to such modification.

Bunny-Wolf World

We use a world derived from Sprague and Ballard (Sprague and Ballard 2003b). In Bunny-Wolf world, our agent is a bunny that must eat food and avoid being eaten by a wolf. The bunny world is a continuing world rather than an episodic world. There is no specified start state and there is no termination of episodes. When the bunny finds and eats food, a new food item appears elsewhere. When the wolf eats the bunny the bunny "respawns" in a new location, similar to video games. We can represent such a bunny agent in our formulation as follows:

- Module 1: FindFood. The bunny agent must find food in order to continue living. When the bunny finds food it gets a reward of 1.0. In each step that it does not eat the bunny gets a reward of -0.1 to represent increasing hunger.
- Module 2: AvoidWolf. The bunny agent must avoid the wolf. Meeting the wolf gives the bunny a reward of -1.0. In each time step that the bunny avoids the wolf the bunny receives a reward of 0.1 to represent that it successfully lived another day.
- Agent's overall goal (implemented in arbitrator): LiveLongProsper – get as much food per time step as possible, which will require balancing food finding with wolf avoidance. The arbitrator's reward function is 0.0 for meeting the wolf, 1.0 for finding food, and 0.5 for each step in which the wolf is avoided but no food is eaten. This is the same as the score used to evaluate algorithm performance (discussed below). Using the score makes sense because maximizing a global "score" is the overall goal of the agent.

To facilitate comparison between Arbi-Q and GM-Sarsa we use a performance metric – a score – that is independent of the reward received by any modules or agents as a whole. The learning of the modules is still guided by their reward functions, but an independent score is necessary for comparison between algorithms to avoid coupling their reward scales. The score we use is 0.0 for meeting the wolf, 1.0 for finding food, and 0.5 for each step in which the wolf is avoided but no food is eaten. We validate Arbi-Q's performance by comparing it with GM-Sarsa, as we discuss below.

We evaluate each algorithm similarly to Sprague and Ballard (Sprague and Ballard 2003b), running each algorithm for n steps, suspending learning every $n/100$ steps to evaluate performance. Performance is evaluated by running the greedy policy in the world for 1000 episodes and calculating the average score per time step. Each algorithm used a discount rate of 0.9 and ϵ-greedy action selection during training with ϵ linearly discounted from 0.4, as in Sprague and Ballard's experiments. For baselines, GM and Arbi-Q algorithms used modules with similarly scaled rewards. For robustness validation, we scaled the AvoidWolf module reward by 10 to simulate the swapping out of separately-authored learning modules. Although we also compared to monolithic, or "flat", reinforcement learners, we do not present a comparison to a flat Q or Sarsa learner here because GM-Sarsa has already been shown to be superior to a flat Q or Sarsa learner on this problem (Sprague and Ballard 2003b).

Results

Empirical results show that the performance of GM-Sarsa degrades when the reward scales of the modules are not comparable. The learning curves depicted in Figure 2 show that GM-Sarsa bunny agent with incomparable reward scales for its modules converges to a lower score than with comparable rewards.

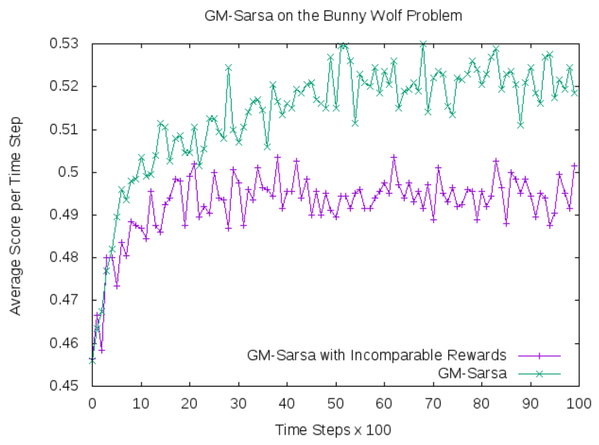

Figure 2: Performance of GM-Sarsa/Q-decomposition on the bunny-wolf problem. The learning curves show that Greatest Mass command arbitration degrades significantly when its module rewards are incomparable.

As Figure 3 shows, Arbi-Q does not exhibit any performance degradation when the agent's modules have incomparable reward scales. Arbi-Q does not use the Q-values of its modules directly. Instead, Arbi-Q learns when it should listen to a particular module. More precisely, Arbi-Q develops a probability distribution for each state which says which module has the best advice in that state. Using the example above with incomparable reward scales, the modules would learn the same local policies using the same Q-values as above, but the arbitrator would learn a policy based on Q-values for selecting modules that chose actions that resulted in particular rewards *for the agent as a whole*. The resulting Q-values for the arbitrator represent *module* selection, not *action* selection. So Arbi-Q learns that when the wolf is close AvoidWolf should decide the bunny agent's action, and when the wolf is comfortably distant FindFood should decide the bunny agent's action.

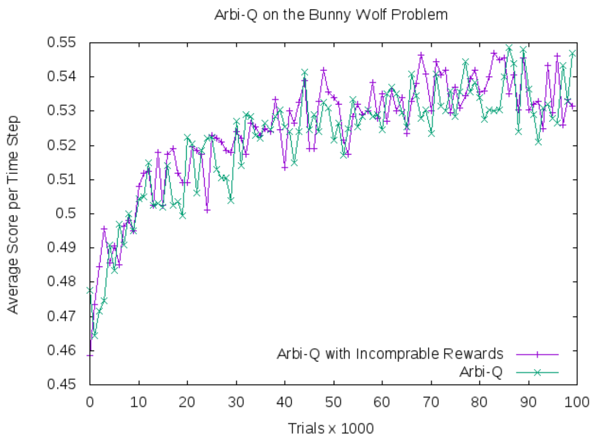

Figure 3: Performance of Arbi-Q on the bunny-wolf problem. Arbi-Q converges to similar scores as GM-Sarsa and shows no degradation in performance when modules have incomparable rewards. By solving the module reward scale comparability problem, Arbi-Q better supports modular reinforcement learning agents with separately written modules.

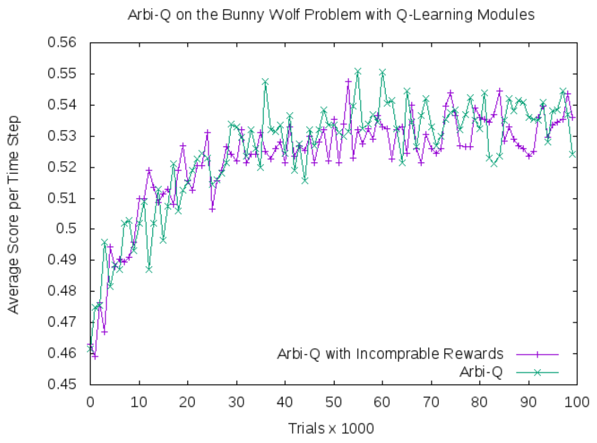

Figure 4: Performance of Arbi-Q on the bunny-wolf problem using Q-learning for the modules and Q-learning to learn the command arbitration policy. Arbi-Q exhibits the same learning performance using Q-learning, suggesting that it is not only robust to modules with incomparable reward scales, but to modules using different algorithms to learn their local policies.

Arbi-Q is Robust to Algorithm Choice

Sprague and Ballard, and Russell and Zimdars showed that Greatest-Mass (Sprague and Ballard 2003b) (or Q-decomposition (Russell and Zimdars 2003)) action selection achieved better learning performance when modules used Sarsa instead of Q-Learning because Sarsa is on-policy and Q-learning is off-policy. The reason for this sensitivity to module learning algorithm is that the arbitrator uses the modules' q-functions directly. Modules using off-policy learning algorithms (Q-learning) update their q-functions based on the optimal future actions for the modules rather than for the agent as a whole, which biases the action selection of an arbitrator using an additive q-function away from the optimal joint policy. Modules using on-policy learning (Sarsa) update their q-functions based on the actual action taken by the agent, resulting in convergence to the optimal global policy as long as the reward scales are comparable (Russell and Zimdars 2003).

Although our primary aim here is to achieve robustness to incomparable reward scales in modules, Arbi-Q is also robust to the algorithm used in the modules. Unlike Greatest-Mass/Q-decomposition, Arbi-Q exhibits the same learning performance whether its modules use Sarsa or Q-Learning, and whether the arbitrator uses Sarsa or Q-learning to learn the delegation policy, as shown in Figure 4.

Discussion

Arbi-Q achieves composability by decoupling module reward scales at the cost of requiring a separately authored reward function. While it is not obvious how to author arbitrator reward, a few rules of thumb have proved useful in practice: (1) thinking of the arbitrator reward function as representing greater good, ”why” find food, ”why” avoid wolf – to live longer; and (2) thinking of the arbitrator reward function as score in a video game, e.g., Pac Man eats food pellets, eats cherries, and either runs from or chases ghosts, but the *reason* for doing these things is to maximize a score.

Each module, as well as the arbitrator, may use state abstraction (a subset of the word state which results in a smaller state space). These state abstractions are often obvious for modules but may may not be obvious for the arbitrator. If the reward function of the command arbitrator uses the original world state rather than a state abstraction, then its state space is the same size as a monolithic reinforcement learner. It would seem that a chief benefit of MRL has been lost. However, in most cases there is still a benefit. In particular, the savings in learning speed compared to a monolithic reinforcement learner is the ratio of actions to the number of modules. The Q-table of a monolithic reinforcement learner would have $|S| \times |A|$ entries. The Q-table of arbitrator has $|S| \times n$ entries where n is the number of modules. So in our bunny-wolf example, with four actions and two modules, even with a naive reward function using no state abstraction the arbitrator's Q-table is half the size of a monolithic reinforcement learner's Q-table. In general, though, greater efficiency can be achieved by using a state abstraction for the arbitrator. Problems for which an arbitrator cannot (easily) be defined with a small enough state space relative to the monolithic state space or where the action set is not significantly larger than the number of modules to yield sufficient improvements in convergence speed may not be good fits for our algorithm.

Related Work

Hierarchical reinforcement learning (HRL) employs a temporal decomposition of the Q function. In the case of HRL, the designer typically specifies a delegation hierarchy of components with points of adaptation where a policy is learned to perform the delegation. The designer programs the policies of some of these components, which constitute a partial specification of the agent's behavior, and some of the components use reinforcement learning to adapt to the hierarchy by learning the control policies for their parts of the problem. The adaptive components relieve the designer from writing the parts of the program that are hard to specify, or require difficult to write adaptivity, and the partial program constrains the learning problem faced by the adaptive components, which speeds convergence. Components can be reused in other contexts, providing for modularity in temporal problem decompositions.

The current state of the art in HRL is based on the theory of **Semi-Markov Decision Processes** (SMDPs). In an SMDP some actions are allowed to take more than one time step. These multi-step actions, sometimes called macros, subroutines, options, or hierarchical machines represent a form of procedural abstraction that may allow a programmer to write reinforcement learning agents in a manner similar to writing other kinds of computer programs.

Precup's Options (Precup, Sutton, and Singh 1998; Sutton, Precup, and Singh 1999; Precup 2000) framework develops a detailed theory of temporal abstraction abstraction based on SMDPs. Dietterich's MAXQ (Dietterich 1998; 2000) and Hierarchical Abstract Machines (Parr and Russell 1998) models decompose an MDP into a hierarchies of smaller MDPs. These smaller MDPs represent *subroutines* in the MAXQ framework or *HAMs* in the *Hierarchical Abstract Machines* framework. Andre's Programmable Hierarchical Abstract Machines (Andre and Russell 2000; 2002) develops a programming language for HAMs called ALisp (Adaptive Lisp), and Marthi and colleagues (Marthi et al. 2005) extend ALisp (Concurrent ALisp) for single reinforcement learning agents with multiple simultaneous actions, such as as robots with multiple effectors or a controller for multiple characters in a computer game. Zhang, Song and Ballard (Zhang, Song, and Ballard 2015) build on Sprague's and Ballard's earlier work on GM-Sarsa to develop three algorithms for deriving a global policy from independent modules. All of these algorithms still rely on the internal Q-values of the modules and thus require comparable reward scales. Rohanimanesh and Mahadevan extended the options HRL framework to concurrent settings in which multiple agents executing multiple simultaneous actions (Rohanimanesh and Mahadevan 2001; 2002). Their work differs from ours in that their framework applies to a single agent taking multiple actions or multiple agents taking simultaneous actions, whereas we are concerned with a single agent executing a single action that is decided by multiple reinforcement learning modules. Marthi and colleagues (Marthi et al. 2005) suggest extending their work in concurrent ALisp to include the Q-decomposition algorithm of Russell and Zimdars (Russell and Zimdars 2003), but this line of research was not pursued. Lau and colleagues developed a MRL system that uses a central coordinator for multiple concurrent MPDs (Lau, Lee, and Hsu 2012). Lau's work differs form ours in that they develop a constraint system in the central coordinator that limits the allowable actions of the component reinforcement learners, thereby constraining their learning. Our approach does not require the arbitrator to know details of component learners, and component learners require no explicit or implicit knowledge of the arbitrator or the other components.

Due to the curse of dimensionality, abstraction of various kinds has long been an active area of research in reinforcement learning. One thread in abstraction is to use examples to guide abstraction. Zang and colleagues used examples of nearly optimal action sequences, or trajectories, to dynamically discover options from data, delivering speedups of up to 30 times in some cases (Zang et al. 2009). Learning from demonstration (Ng, Russell, and others 2000; Zang et al. 2010b) uses human input to improve reinforcement learning performance. Zang and colleagues developed a value function approximation algorithm that leveraged human input to speed convergence for function approximation-based reinforcement learning algorithms (Zang et al. 2010a). Cobo Rus and colleagues' Abstraction from Demonstration technique uses human demonstrations to infer state abstractions and builds policies based on those state abstractions (Rus et al. 2011; Rus, Isbell, and Thomaz 2012; Rus et al. 2014).

Another thread in abstraction seeks to use models from the physical world to create abstractions of (simulated) physical state spaces. Cobo Rus and colleagues created abstractions of state spaces by organizing state spaces into classes of objects and using non-optimal Q-functions to estimate the risk of ignoring certain classes of objects. Cobo Rus's Object-Focused Q-Learning (OFQ) achieved exponential speedups in some cases (Rus, Isbell, and Thomaz 2013). Scholz and colleagues developed Physics-Based Reinforcement Learning (Scholz et al. 2014), which uses computational physics engines such as Box2D (Catto 2013) as model representations, resulting in more sample-efficient learning compared to traditional object-oriented MDP approaches. Physics-based reinforcement learning was then applied successfully in robotic mobile manipulation (Scholz et al. 2015) and robot navigation (Scholz et al. 2016) applications.

Contributions, Limitations, and Future Work

The primary contribution of our reformulation of MRL and the Arbi-Q command arbitration algorithm is the reusability afforded by reward scale decoupling. Our algorithm makes it possible for modules written by different programmers with different reward scales to be used together within the same MRL agent. So our MRL algorithm supports software engineering, in which it is important not only to decompose problems into sub-problems, but to be able to compose new solutions from solutions to separately authored sub-problems. This property is important to us because we are integrating reinforcement learning into a programming language. We have implemented an experimental language and conducted programmer studies with promising results

(Simpkins, Rugaber, and Isbell 2017) that suggest continued development. Nevertheless, the present work on modular reinforcement learning has limitations.

The first limitation of Arbi-Q is that the agent's action always comes from one of the modules. It may be that the best action in a given state is not chosen by any module. Approaches to module cooperation such as negotiated-W learning (Humphrys 1997) have been shown to be inferior to GM-Sarsa (and thus Arbi-Q) for problems such as the one presented here, but it is worth investigating mixed approaches to action selection. The recent work of Zhang, Song and Ballard (Zhang, Song, and Ballard 2015) may provide a promising direction.

The second limitation is that achieving a substantial reduction in total state space depends on the state abstraction of the arbitrator, which depends on the agent-level reward function. Techniques for authoring arbitrator reward functions need to be investigated.

A third limitation is that we use fairly standard tabular Q-learning algorithms. Future work should investigate newer function approximation approaches such as hierarchical Deep Q-Networks (Kulkarni et al. 2016).

Finally, we have demonstrated our algorithm's performance on a small domain chosen so that we can compare it directly to previous work. Future work should evaluate our approach on larger domains with varied characteristics.

References

Aissani, N.; Beldjilali, B.; and Trentesaux, D. 2009. Dynamic scheduling of maintenance tasks in the petroleum industry: A reinforcement approach. *Engineering Applications of Artificial Intelligence* 22(7):1089–1103.

Andre, D., and Russell, S. 2000. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13. MIT Press.

Andre, D., and Russell, S. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI-02*. Edmonton, Alberta: AAAI Press.

Arrow, K. J. 1963. *Social Choice and Individual Values*. John Wiley and Sons, 2nd edition.

Bhat, S.; Isbell, C.; and Mateas, M. 2006. On the difficulty of modular reinforcement learning for real-world partial programming. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*.

Catto, E. 2013. Box2d physics engine.

Chaari, T.; Chaabane, S.; Aissani, N.; and Trentesaux, D. 2014. Scheduling under uncertainty: Survey and research directions. In *Advanced Logistics and Transport (ICALT), 2014 International Conference on*, 229–234. IEEE.

Dietterich, T. G. 1998. The MAXQ method for hierarchical reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, 118–126. Morgan Kaufmann, San Francisco, CA.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Humphrys, M. 1997. Action selection methods using reinforcement learning. Technical Report UCAM-CL-TR-426, University of Cambridge, Computer Laboratory.

Konidaris, G., and Barto, A. 2006. An adaptive robot motivational system. In *International Conference on Simulation of Adaptive Behavior*, 346–356. Springer.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 3675–3683.

Lau, Q. P.; Lee, M. L.; and Hsu, W. 2012. Coordination guided reinforcement learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 215–222. International Foundation for Autonomous Agents and Multiagent Systems.

Marthi, B.; Russell, S.; Latham, D.; and Guestrin, C. 2005. Concurrent hierarchical reinforcement learning. In *IN PROCEEDINGS IJCAI-2005*, 779–785.

Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, 663–670.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In Jordan, M. I.; Kearns, M. J.; and Solla, S. A., eds., *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

Precup, D.; Sutton, R. S.; and Singh, S. 1998. *Theoretical results on reinforcement learning with temporally abstract options*. Berlin, Heidelberg: Springer Berlin Heidelberg. 382–393.

Precup, D. 2000. *Temporal Abstraction in Reinforcement Learning*. Ph.D. Dissertation, University of Massacheusetts Amherst.

Roberts, K. W. 1980. Interpersonal comparability and social choice theory. *The Review of Economic Studies* 421–439.

Rohanimanesh, K., and Mahadevan, S. 2001. Decision-theoretic planning with concurrent temporally extended actions. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 472–479. Morgan Kaufmann Publishers Inc.

Rohanimanesh, K., and Mahadevan, S. 2002. Learning to take concurrent actions. In *Advances in neural information processing systems*, 1619–1626.

Rowe, J. P., and Lester, J. C. 2013. A modular reinforcement learning framework for interactive narrative planning. In *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 57–63.

Rummery, G. A., and Niranjan, M. 1994. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering.

Rus, L. C. C.; Zang, P.; Isbell, C. L.; and Thomaz, A. 2011. Automatic State Abstraction from Demonstration. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*.

Rus, L. C. C.; Subramanian, K.; Isbell, C. L.; Lanterman, A.; and Thomaz, A. 2014. Abstraction from Demonstration for Efficient Reinforcement Learning in High-Dimensional Domains. *Artificial Intelligence* 216(0):103–128.

Rus, L. C. C.; Isbell, C. L.; and Thomaz, A. 2012. Automatic Decomposition and State Abstraction from Demonstration. In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Rus, L. C. C.; Isbell, C. L.; and Thomaz, A. 2013. Object Focused Q-learning for Autonomous Agents. In *Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Russell, S., and Zimdars, A. L. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*.

Scholz, J.; Levihn, M.; Isbell, C. L.; and Wingate, D. 2014. A Physics-Based Model Prior for Object-Oriented MDPs. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Scholz, J.; Levihn, M.; Isbell, C. L.; Christensen, H.; and Stilman, M. 2015. Learning Non-Holonomic Object Models for Mobile Manipulation. In *Proceedings of the the 2015 IEEE International Conference on Robotics and Automation (ICRA)*.

Scholz, J.; Nehchal, J.; Levihn, M.; and Isbell, C. L. 2016. Navigation Among Movable Obstacles with Learned Dynamic Constraints. In *Proceedings of the the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Simpkins, C.; Bhat, S.; Isbell, C.; and Mateas, M. 2008. Towards adaptive programming: Integrating reinforcement learning into a programming language. In *OOPSLA '08: ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Onward! Track*.

Simpkins, C.; Rugaber, S.; and Isbell, C. L. 2017. Dsl design for reinforcement learning agents. In *Workshop on Domain-Specific Language Design and Implementation (DSLDI)*.

Singh, S.; Jaakkola, T.; Littman, M. L.; and Szepesvári, C. 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning* 38(3):287–308.

Sprague, N., and Ballard, D. 2003a. Eye movements for reward maximization. In *Advances in neural information processing systems*, None.

Sprague, N., and Ballard, D. 2003b. Multiple-goal reinforcement learning with modular sarsa(o). In *IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence*, 1445–1447. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sprague, N.; Ballard, D.; and Robinson, A. 2007. Modeling embodied visual behaviors. *ACM Transactions on Applied Perception (TAP)* 4(2):11.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

Zang, P.; Zhou, P.; Minnen, D.; and Isbell, C. L. 2009. Discovering Options from Example Trajectories. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML)*.

Zang, P.; Irani, A.; Zhou, P.; Isbell, C. L.; and Thomaz, A. 2010a. Using Training Regimens to Teach Expanding Function Approximators. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Zang, P.; Tian, R.; Isbell, C. L.; and Thomaz, A. 2010b. Batch versus interactive learning by demonstration. In *Proceedings of the Ninth International Conference on Development and Learning (ICDL)*.

Zhang, R.; Song, Z.; and Ballard, D. H. 2015. Global policy construction in modular reinforcement learning. In *AAAI*, 4226–4227.