On the Difficulty of Modular Reinforcement Learning for Real-World Partial Programming

Sooraj Bhat, Charles L. Isbell Jr., Michael Mateas

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
{sooraj, isbell, michaelm}@cc.gatech.edu

Abstract

In recent years there has been a great deal of interest in "modular reinforcement learning" (MRL). Typically, problems are decomposed into concurrent subgoals, allowing increased scalability and state abstraction. An arbitrator combines the subagents' preferences to select an action. In this work, we contrast treating an MRL agent as a set of subagents with the same goal with treating an MRL agent as a set of subagents who may have different, possibly conflicting goals. We argue that the latter is a more realistic description of real-world problems, especially when building partial programs. We address a range of algorithms for single-goal MRL, and leveraging social choice theory, we present an impossibility result for applications of such algorithms to multi-goal MRL. We suggest an alternative formulation of arbitration as scheduling that avoids the assumptions of comparability of preference that are implicit in single-goal MRL. A notable feature of this formulation is the explicit codification of the tradeoffs between the subproblems. Finally, we introduce A^2BL, a language that encapsulates many of these ideas.

Introduction

Modular reinforcement learning (MRL), as we will use the phrase, refers to the decomposition of a complex, multi-goal problem into a collection of simultaneously running single-goal learning processes, typically modeled as Markov Decision Processes. Typically, these subagents share an action set but have their own reward signal and state space. At each time step, every subagent reports a numerical preference for each available action to an arbitrator, which then selects one of the actions for the agent as a whole to take. This general approach is an attempt to avoid the drawbacks of simpler arbitration techniques such as *command arbitration*, in which subagents each only recommend a single action, or *command fusion*, in which the arbitrator executes some type of "combination" action, such as an average over the actions suggested by the subagents.

One natural way to represent these numerical preferences is to simply use Q-values learned by each component MDP. Optimally combining subagent Q-values in a meaningful way has thus become the focus of recent work. The argument is that even though the subagents may be pursuing conflicting subgoals, they are each necessary components of an enclosing goal, so we would like to maximize the expected reward for each subagent as much as possible. Arbitration techniques differ in how they try to accomplish this maximization. For example, one approach is to select an action to maximize average happiness, *i.e.* the action maximizing $\sum_j Q_j(s, a)$. Another is to adopt a winner-take-all strategy and allow the subagent with the largest Q-value for any action to select the action, *i.e.* the action maximizing $\max_j Q_j(s, a)$.

Although sometimes implicit, all of these arbitration approaches assume the subagent reward signals are comparable. We argue that this is not the general case for real-world, multiple-goal problems. In practice, it is rare that a "ground truth" reward signal is available; instead, we as problem solvers are the ones designing the reward signal. The reward signal design process for a single-goal problem may be tricky (as the designer may only have a general idea of how reinforcing feedback should come from the environment) but is usually manageable as there is only a single well-defined goal in mind. The multiple-goal case, however, is more complicated: not only must the designer properly craft a reward signal for each subagent, she must also ensure that the reward units are consistent between the subproblems. This is a significant, and perhaps impossible, burden on the designer, especially as the number of subproblems—and thus the size of the software problem facing the designer—increases.

In light of this comparability problem, we show in this paper that it is impossible to construct an arbitration function that satisfies a few basic, yet desirable, properties. We obtain the result by leveraging *social choice theory*. In particular, we reduce the problem of constructing an arbitration function to a variant of Arrow's Impossibility Theorem for social ordering functions (Arrow 1966). Characterizing MRL as a social welfare problem opens up new avenues of analysis, as we can now start applying established results from the social choice literature to MRL.

The paper is organized as follows. First we motivate our discussion with an in-depth explanation of the problem of subagent reward incomparability. In particular we take the view of decomposition as one of *partial programming*. Next, we introduce some necessary formalism and background before presenting the impossibility result. Finally,

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

we propose a mechanism for addressing the social choice dilemma and describe A^{2}BL, a programming language that makes that mechanism concrete.

Motivation

Modular Reinforcement Learning arises in a variety of contexts. In our own work, it arises in efforts to build a system for partial programming; that is, a designer or programmer specifies only that part of the program known to be correct, allowing a learning system to learn the rest from experience, typically using reinforcement learning (alternatively, one can think of partial programming as a way for a designer to inject prior knowledge into a learning agent). Current approaches to incorporating adaptivity into programming languages have focused on allowing the programmer to constrain the set of policies considered by hand-authoring a subroutine hierarchy (Andre & Russell 2000; Dietterich 1998). This framework is commonly referred to as *hierarchical reinforcement learning* (HRL) and corresponds to a temporal decomposition of goals. For partial programming, one would also like to support the concurrent subgoal decomposition offered by MRL.

In practice, we have found that while it is usually reasonable to expect a programmer to design a reward signal for a single goal, it is much harder to maintain consistency in the presence of multiple goals. Consider the `Predator-Food` grid world. In this world, there are two main activities: avoiding the predator and finding food. At every time step, the agent must pick a direction to move. Food appears randomly at fixed locations, and there is a predator in the environment who moves towards the agent once every other time step. This learning problem is naturally described in the MRL framework: the two concurrent subgoals are `FindFood` and `AvoidPredator`. In this example, it is somewhat straightforward to design a reward signal for each subgoal: in `FindFood`, one can assign a large positive reward when the agent lands on a grid square with food in it; in `AvoidPredator`, one can assign a large negative reward when the agent is captured by the predator. However, how do we assign the magnitudes of the rewards, particularly in relation to each other? How can we know that +10 units of reward in the `FindFood` subgoal is actually equivalent to +10 units of reward in the `AvoidPredator` subgoal? It is especially difficult to answer these questions because there are no objective criteria for determining the answer.

If keeping the reward signal consistent between subproblems is a difficult task for one programmer, it becomes a nightmare in the presence of multiple programmers. We know that developers using an adaptive programming language will have to solve the same multi-programmer coordination and code reuse problems faced by practitioners using standard languages. This is especially true in large, real-world systems. So, if we require that subagents are designed in such a way that rewards must be consistent across subagents, any reasonably-sized application will require multiple programmers to maintain reward consistency across multiple, independently written subgoals, in addition to all the standard multi-programmer coordination issues.

The requirement of maintaining reward consistency also inhibits software reuse. Suppose, for instance, we have managed to design the reward signals for `FindFood` and `AvoidPredator` for an Environment A, such that reward consistency is maintained. Now assume that there is an Environment B that can make use of the `FindFood` subgoal, but does not need `AvoidPredator`. Ideally, we would like to enjoy the benefits of software reuse and simply invoke the prewritten `FindFood` subgoal; however, we cannot naively do so as the reward signal for `FindFood` has been tuned for Environment A. The design of `FindFood` has involved tradeoffs with `AvoidPredator`. So now, we must instead redesign the reward signal for `FindFood`, this time for Environment B. On the other hand, if we only require the reward signal to be *internally* consistent, rather than consistent across different subgoals, it is not only easier for the programmer to design, but also easier to reuse.

Requiring the programmer to keep the reward signal consistent across multiple subgoals is therefore an unreasonable requirement. This being the case, we cannot treat the subagent reward signals (and thus Q-values) as comparable. Once cast this way, it becomes useful to think of our problem in terms of social choice theory. Social choice theory is a branch of economics where the main goal is to find ways to combine the preferences for a set of social outcomes (*e.g.*, over election candidates, or propositions) held by a population of individuals. As we shall see below, results from social choice theory severely constrain our ability to address problems in MRL.

Note that we are viewing the problem as the composition of complex systems of multiple decision makers, versus the standard reinforcement learning view of a single monolithic decision maker. Our approach extends and generalizes current approaches to compositional reinforcement learning, which still depend on a global reward signal as a common thread of global knowledge running through all parts of the system. Furthermore, this characterization lends itself to other types of coordination-of-learning problems, from high-level systems such as multiagent robot systems to low-level systems like collections of modular learning components in an agent's mind.

Background

In the MRL framework, a learning agent, M, is represented by a set of n subproblems (subagents), $M = \{M_j\}_1^n$, each having its own reward signal and state space. The typical formulation is to have subagents share an action set, thus $M_j = (S_j, A, R_j)$, but we will relax that assumption later on. The agent can be thought of as having a state space that is a cross product of the subagent state spaces, $S = S_1 \times S_2 \times \ldots \times S_n$. Traditionally, it is assumed that the agent's reward is a sum of subagent rewards, $R(s,a) = \sum_{j \in N} R_j(s,a)$, where $N = \{1, \ldots, n\}$. Thus, M is well-defined as a reinforcement learning problem, $M = (S, A, R)$, and the goal is to find a joint policy $\pi(s)$, composed of locally learned policies $\pi_j(s)$, that is optimal for the whole agent. We let $Q_j(s,a)$ refer to the Q-value learned by subagent j for state s and action a. Again,

traditionally, $Q(s,a) = \sum_{j \in N} Q_j(s,a)$.

Once we disallow the assumption that subagent rewards are comparable, we cannot assume $R(s,a) = \sum_{j \in N} R_j(s,a)$. In this case, M is no longer well-defined, and so we must change our notion of what it means to solve M optimally. Our new notion of optimality will be in the form of requiring certain properties of the arbitration function. These properties are presented in detail in the next section.

Existing arbitration techniques

To illustrate the role of the arbitration function, we provide here a brief overview of some existing arbitration techniques. A more in-depth overview is available in (Sprague & Ballard 2003), from which we also borrow some nomenclature.

Greatest Mass Q-Learning (GM-Q). An arbitration function consistent with GM-Q orders actions by their summed Q-value, $X_a = \sum_j Q_j(s,a)$. The arbitrator then selects the action with the largest X_a value (the one with the "greatest mass"). This approach can be viewed as an attempt to maximize average happiness.

Top Q-Learning (Top-Q). Humphrys (1996) observes that selecting an action according to GM-Q may not be particularly good for any single agent. An alternative is to instead order the actions by their top Q-value, $X_a = \max_j Q_j(s,a)$. This is a style of winner-take-all arbitration, as the subagent who reported the largest Q-value effectively chooses the action.

Negotiated W-Learning (Neg-W). Top-Q suffers from the drawback that the subagent with the top Q-value may not have a strong preference between actions. Humphrys suggests an alternative for action selection, called negotiated W-learning, in which the subagent with the most to lose is allowed to select the action.

Ideal Arbitration

Let us consider properties that we might wish an ideal arbitrator to possess. First, we define some notation.

Let us assume the current state s is fixed, as the state remains unchanged during the invocation of the arbitration function. At a given time step, let $U \in \mathbf{R}^{m \times n}$ be a matrix of values containing the subagent preferences, with $U_{a,j}$ representing the numerical preference subagent j has for action a, and $m = |A|$ the number of actions. Thus, $U_{a,j} = Q_j(s,a)$. Let $\mathcal{A}$ be the set of all possible total orderings of the elements of A.

An arbitration function f is a function from $\mathcal{D} \subseteq \mathbf{R}^{m \times n}$ to $\mathcal{A}$. In other words, f takes an assignment of action preferences by each subagent and generates a social preference ordering over all the actions. In our case, "society" is the agent, and the top-ranked action is the best action for the agent as a whole to take.

Property 1 (Universality) $\mathcal{D} = \mathbf{R}^{m \times n}$.

This property requires the domain of f to be universal, *i.e.* f must be able to handle all possible inputs. Worlds can be constructed to generate all possible settings of the $Q_j(s,a)$, thus all matrices $U \in \mathbf{R}^{m \times n}$ are attainable. We want an arbitration function that can handle all these cases.

Property 2 (Unanimity) *For any pair $a, b \in A$ and for all $U \in \mathcal{D}$, if $[\forall j \in N,\ U_{a,j} > U_{b,j}]$, then $f(U)$ ranks a as strictly preferred to b.*

This property states that if every subagent prefers action a over action b, then the arbitration function should agree with the unanimous opinion and output an ordering that ranks a over b. This is also commonly known as Pareto Efficiency.

Property 3 (Independence of Irrelevant Alternatives) *For any $U, U' \in \mathcal{D}$ and subset of actions $B \subseteq A$, if $[\forall b \in B, \forall j \in N,\ U_{b,j} = U'_{b,j}]$, then $f(U)$ and $f(U')$ rank the elements of B in the same order.*

This is a requirement that the relative ordering of a subset of actions should only depend on the numerical preferences for those actions. Another interpretation is that the presence or absence of an irrelevant alternative $x \in A$ should not affect a subagent's preference between two actions $a, b \in A \setminus \{x\}$. Arbitration functions which obey this condition are also *strategy-proof*: a subagent cannot raise the rank of an action it prefers by dishonestly reporting its numerical preferences. This effectively prevents subagents from inadvertently "gaming the system", *i.e.* from learning the value of *proposing* an action in order to achieve better reward. This is desirable from the point of view of software reuse: a subgoal used under one parent goal can also be used under a second parent goal without fearing that the subgoal has only learned to propose actions in the presence of the first parent, without actually learning the true utility of the actions.

Property 4 (Scale Invariance) *For $U, U' \in \mathcal{D}$, if $[\forall j \in N$ there exists $\alpha_j, \beta_j \in \mathbf{R}$ (with $\beta_j > 0$) such that $U'_{a,j} = \alpha_j + \beta_j \cdot U_{a,j}]$, then $f(U) = f(U')$.*

This property is the formal counterpart to our reward incomparability assertion. It states that a positive affine transformation of a subagent's Q-values should not affect the ordering generated by the arbitration function. GM-Q, Top-Q and Neg-W all violate this property—under these schemes, an agent can multiply its Q-values by a large constant and completely determine the selected action. This condition essentially forces f to treat the numerical preferences ordinally.

Note that requiring scale invariance allows for a more applicable theorem than requiring invariance under *arbitrary* monotonic transformations. Scale invariance is actually a weaker condition.

Property 5 (Non-Dictatorship) *Let $R_j \in \mathcal{A}$ be the ordering over actions for subagent j. For all $U \in \mathcal{D}$ there does not exist $j \in N$ such that $f(U) = R_j$.*

This property requires that there should not exist a privileged subagent whose preferences completely determine the output of the arbitration function. On its face, this is desirable: decomposing a problem just to have a single subagent

be the only one who has a say defeats the purpose of the decomposition.

Theorem 1 *If $|A| \geq 3$, then there does not exist an arbitration function that satisfies Properties 1–5.*

PROOF: At this point we have reduced the MRL framework to the same situation addressed by (Roberts 1980). Roberts' proof is essentially a refinement of Arrow's Impossibility Theorem (1966), with the addition of invariance classes, such as scale invariance, to map numeric preferences to symbolic preferences. The full details of this proof are beyond the scope of this paper; however, we refer the interested reader to (Roberts 1980) for a complete treatment.

Note that we have formulated the problem so that we only need to select a single best action (social *choice*), while the theorem is for total orderings (social *ordering*). The problems turn out to be equivalent. It is possible to prove the choice version of the theorem from the ordering version, and it has been shown that the logical underpinnings of the two are identical (Reny 2000). The theorem also extends to the case where the action sets available to each agent are not identical (but may overlap), *i.e.* $A_i \neq A_j$ for $i \neq j$ (Doyle & Wellman 1991). The case where individual agents have different actions arises naturally for MRL in the partial programming context, as it is not uncommon that different subgoals may use different actions to achieve their task.

Scheduling as Arbitration

These properties define a class of arbitration functions that behave rationally and attempt to please all of its constituents, based *only* on the numerical preferences it sees from the subagents, disregarding any type of external information or criteria, such as fairness concerns or any knowledge on how to compare the subagent preferences. As we have shown, this task is impossible in its full generality. Presumably, if we did have some extra knowledge, we would like to incorporate it, perhaps making the problem feasible.

It is useful to examine where knowledge regarding the learning problem resides. We can locate the global knowledge regarding the learning problem either: (1) completely at the arbitrator, forgoing the benefits of problem decomposition and requiring a globally-crafted reward signal; (2) completely at the subagents, with the containing goal being merely a mindless arbitration rule with no knowledge about tradeoffs or any other domain knowledge; or (3) a mixture in which the subagents have enough knowledge to accomplish their own task (a local view, if you will), and the containing goal contains enough information to arbitrate properly, either as a statically-written or dynamically-learned rule.

Options (1) and (2) are infeasible. In particular, we have shown (2) to be impossible. Previous approaches are able to "successfully" accomplish (2) because they implicitly require a globally crafted reward signal, putting us in option (1).

Thus, we need to pursue the "smarter arbitrator" option. We claim it is easier to tie in information relating the subgoals in the arbitration function rather than in the reward signal. Furthermore, this makes tradeoffs explicit. For instance, under the globally crafted reward signal regime, if the reward signal somewhere inside a subgoal had been reduced from +10 to +5 for some reason—for instance, to deemphasize its role in relation to another subagent—the programmer would have to document this reason so that future maintainers of the code will know why the numbers are the way they are (this will be especially true of large codebases). In our scheme, that tradeoff would be codified in the arbitration function. This improves code quality in the same way assertions codify invariants or Makefiles codify build processes.

We use the term *scheduler* to refer to these smarter arbitrators, in loose analogy with schedulers for operating systems, to hint at how they use information outside of what the subagents are reporting to inform their arbitration decisions. Just as a thread scheduler might use information about processor usage to fairly divide up cycles, an arbitration scheduler might incorporate subagent reward comparability into its arbitration decision. For example, if one subagent receives reward in euros, and another subagent receives reward in dollars, then the tradeoff between the subagent rewards is known exactly: the rewards are comparable after a simple exchange rate calculation.

The reader may wonder how we have circumvented the impossibility result. In a sense, a scheduler acts as a benevolent dictator. In particular, it encapsulates a "selfish" idea of what is important to a larger goal that need not take fairness or the happiness of each subagent into account. More to the point, in general, it is an agent with its own agenda and a repository for any global domain knowledge the programmer wants to inject into the program solution. Further, the scheduler can now be modeled as just another agent—albeit a very important one—capable of its own goals and adaptation.

The social choice literature explores a variety of approaches for circumventing the impossibility result, typically through various relaxations of either Property 2 or 3. By placing MRL in a social choice framework, this body of results is now potentially applicable to MRL. Our solution to scheduling can be viewed as a relaxation of the non-dictatorship property (Property 5): we have a benevolent dictator who is constrained to select among the actions available to its subagents. While this relaxation is non-standard in the social choice literature, in the context of combining human-authored and learned knowledge, a benevolent dictator is a natural point at which to inject domain knowledge.

A Situated Example

In order to demonstrate some of the points we have made, it is useful to examine a simple, concrete learning problem in a partial programming context. We will consider the `Predator-Food` world, as would be implemented in An Adaptive Behavior Language (A^2BL). A^2BL is a derivative of A Behavior Language (ABL), a reactive planning language with Java-like syntax based on the Oz Project believable agent language Hap (Bates, Loyall, & Reilly 1992). An ABL agent consists of a library of sequential and parallel behaviors with reactive annotations. Each behavior consists of

a set of steps to be executed either sequentially or in parallel. There are four basic step types: acts, subgoals, mental acts and waits. Act steps perform an action in the world; subgoal steps establish goals that must be accomplished in order to accomplish the enclosing behavior; mental acts perform bits of pure computation, such as mathematical computations or modifications to working memory; and wait steps can be combined with continually-monitored tests to produce behaviors that wait for a specific condition to be true before continuing or completing.

The agent dynamically selects behaviors to accomplish specific goals and attempts to instantiate alternate behaviors to accomplish a subgoal whenever a behavior fails. The current execution state of the agent is captured by the active behavior tree (ABT) and working memory. Working memory contains any information the agent needs to monitor, organized as a collection of working memory elements (WMEs). There are several one-shot and continually-monitored tests available for annotating a behavior specification. For instance, preconditions can be written to define states of the world in which a behavior is applicable. These tests use pattern matching semantics over working memory familiar from production rule languages; we will refer to them as *WME tests*.

We have extended some of these concepts for A^{2}BL. The most notable addition is the `adaptive` keyword, used as a modifier for behaviors. When modifying a sequential behavior, `adaptive` signifies that, instead of pursuing the steps in sequential order, the behavior should learn a policy for which step to pursue, as a function of the state of the world. As there could be a large amount of information in working memory (which is the agent's perception of the state of the world), we have introduced a `state` construct to allow the programmer to specify which parts of working memory the behavior should pay attention to in order to learn an effective policy. This allows for human-authored state abstraction.

Behaviors normally succeed when all their steps succeed. Because it is unknown which steps the policy will ultimately execute, adaptive behaviors introduce a new continually-monitored condition, the `success_condition`, which encodes the behavior's goal. When the success condition becomes true, the behavior immediately succeeds. Finally, to learn a policy at all, the behavior needs a reinforcement signal. With the `reward` construct, authors specify a function that maps world state to such a reinforcement signal. In natural analogy to existing ABL constructs, these new constructs each make use of WME tests for reasoning and computing over working memory.

An adaptive collection behavior is specifically designed for modeling the concurrency of MRL. This type of behavior contains within it several adaptive sequential behaviors, which correspond to the subagents in the MRL framework.

An agent composed of a single adaptive sequential behavior and no other behaviors is isomorphic to the traditional, non-decomposed view of reinforcement learning; the usual methods can be used to learn the Q-function. An agent composed of only sequential behaviors and adaptive sequential behaviors is isomorphic to an agent decomposed according to the HRL framework. When the underlying

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    // see text for arbitration possibilities
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY) (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }

  // subgoal 2
  adaptive sequential behavior AvoidPredator() {
    reward {
      -10 if { (PredatorWME) }
    }
    state {
      (PredatorWME x::predX y::predY) (SelfWME x::myX y::myY)
      return (myX,myY,predX,predY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }

  // ...
}
```

Figure 1: An A^{2}BL agent for the `Predator-Food` world.

world is an MDP, the learning problem is equivalent to a Semi-Markov Decision Process (SMDP), in which actions may take multiple time steps to complete. Methods employing a temporal decomposition of the Q-function have been used to efficiently learn a policy for such partial programs (Dietterich 1998; Andre & Russell 2000). An agent composed of a single adaptive collection behavior is isomorphic to an agent decomposed according to the MRL framework. With the assumption that subagent rewards are comparable, this corresponds to a concurrent decomposition of the Q-function. It is known that subagents must use an on-policy learning method in order to learn Q-values that are not overly optimistic, so that GM-Q converges to the globally optimal policy (Russell & Zimdars 2003; Sprague & Ballard 2003). Typical ABL programs, however, consist of a mixture of sequentially and concurrently running components, a style of programming we would like to continue to support. This requires a deeper understanding of more complex, "mixture" decompositions of the Q-function; this is the subject of future work.

Now consider the sample code for the `Predator-Food` world in Figure 1. Note that the rewards in each subgoal are not comparable: the +100 in `FindFood` cannot be compared directly to the -10 in `AvoidPredator`. Though this complicates the arbitration, it allows the code for the subgoals to be reusable, either by the invocation of the subgoal from a different context, or even when the source code for the subgoal is copied into another file altogether. Furthermore, because the rewards for a subgoal do not have to be

changed, it is also easier to reuse persistent representations of learned policy (Q-values saved to a file, for example).

Now that we've defined the two adaptive subgoals, we now need to define an arbitration function on the enclosing goal, `LiveLongProsper`. As we have shown earlier, it is impossible to construct an ideal arbitration function, so we cannot employ the compiler to automatically generate an all-purpose arbitration rule. Instead, the programmer must define an arbitration function, either hand-authored or learned.

A hand-authored arbitration function encodes the tradeoffs the programmer believes to be true about the subagent Q-values. In this example, we may decide that the benefit of finding food equals the cost of running into a predator; given our reward signals, the arbitrator would select the action maximizing $\frac{1}{10}Q_1(s,a) + Q_2(s,a)$. Alternatively, the hand-authored arbitration function could be independent of the subagent Q-values; to simply avoid starvation, for instance, one might consider round-robin scheduling.

Finally, we could try posing `LiveLongProsper`'s arbitration task as another reinforcement learning task, with its own reward function encapsulating a notion of goodness for living well, as opposed to one that only makes sense for finding food or avoiding a predator. For example, the reward function might provide positive feedback for having more offspring; this would be an "evolutionary" notion of reward.

The reader may wonder why `FindFood` and `AvoidPredator` should have their own reward signals if one is available for `LiveLongProsper`. The reasons should be familiar: modularity and speed of learning. The reward signal for `FindFood`, for instance, is specifically tailored for the task of finding food, so the learning should converge more quickly than learning via an "indirect" global reward signal. Further, with the right state features, the behavior should be reusable in different contexts. Specifying a reward signal for each behavior allows the reward signals to embody what each behavior truly cares about: `FindFood` cares about finding gridsquares with food, `AvoidPredator` cares about avoiding the predator, and `LiveLongProsper` cares about ensuring the future of the `FurryCreature` race.

Conclusion

Casting the modular reinforcement learning framework in the light of social choice theory opens up new avenues for analysis by bringing in many established results from social choice theory and the study of voting mechanisms into the analysis of MRL.

In this work we have advocated the codification of tradeoffs between MRL subproblems in the form of a human-authored (and possibly adaptive) arbitration function, which we call a scheduler. We highlight the problem of subagent reward incomparability, leading us to the result that it is impossible to construct an ideal arbitration function without injecting external domain knowledge.

At this point, we have defined a language, A^{2}BL, that allows us to place the tradeoffs inherent in scheduling in programming modules in such a way that makes those tradeoffs more transparent and thus easier for programmers to reason about. We believe this approach better supports code reuse and partial programming, especially in large, multi-programmer projects.

Acknowledgments

We would like to thank Michael Holmes for suggesting that an impossibility theorem would clarify the tradeoffs in the composition of independent learners, as well as an anonymous reviewer who provided us with many interesting perspectives on this work. We acknowledge the support of DARPA under contract No. HR0011-04-1-0049.

References

Andre, D., and Russell, S. 2000. Programmable Reinforcment Learning Agents. In *Advances in Neural Information Processing Systems*.

Arrow, K. J. 1966. *Social Choice and Individual Values*. John Wiley & Sons, 2nd edition.

Bates, J.; Loyall, A. B.; and Reilly, W. 1992. Integrating Reactivity, Goals, and Emotion in a Broad Agent. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*.

Dietterich, T. G. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*.

Doyle, J., and Wellman, M. P. 1991. Impediments to Universal Preference-Based Default Theories. *Artificial Intelligence* 49(1–3):97–128.

Humphrys, M. 1996. Action Selection Methods using Reinforcement Learning. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 135–144.

Karlsson, J. 1997. *Learning to Solve Multiple Goals*. Ph.D. Dissertation, University of Rochester.

Mateas, M., and Stern, A. 2002. A Behavior Language for Story-Based Believable Agents. *IEEE Intelligent Systems* 17(4):39–47.

Reny, P. J. 2000. Arrow's Theorem and the Gibbard-Satterthwaite Theorem: A Unified Approach. http://home.uchicago.edu/ preny/papers/arrow-gibbard-satterthwaite.pdf. Web article.

Roberts, K. W. S. 1980. Interpersonal Comparability and Social Choice Theory. In *The Review of Economic Studies*, 421–439.

Russell, S., and Zimdars, A. L. 2003. Q-Decomposition for Reinforcement Learning Agents. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*.

Sprague, N., and Ballard, D. 2003. Multiple-Goal Reinforcement Learning with Modular Sarsa(0). In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Workshop paper.