Combining Model-Based Meta-Reasoning and Reinforcement Learning For Adapting Game-Playing Agents

Patrick Ulam, Joshua Jones, and Ashok Goel
College of Computing, Georgia Institute of Technology
Atlanta, USA 30332

Abstract

Human experience with interactive games will be enhanced if the software agents that play the game learn from their failures. Techniques such as reinforcement learning provide one way in which these agents may learn from their failures. Model-based meta-reasoning, a technique in which an agent uses a self-model for blame assignment, provides another. This paper evaluates a framework in which both these approaches are combined. We describe an experimental investigation of a specific task (defending a city) in a computer war strategy game called FreeCiv. Our results indicate that in the task examined, model-based meta-reasoning coupled with reinforcement learning enables the agent to learn the task with performance matching that of an expert designed agent and with speed exceeding that of a pure reinforcement learning agent.

Introduction

Intelligent agents acting as non-player characters (NPC) in interactive games often fail in their tasks. However, NPC's in most commercially available interactive games generally do not learn from these failures. As a result, the human player may tire of playing the game. The human player's experience with an interactive game surely will be enhanced if these agents learned from their failures and minimize their recurrence.

Meta-reasoning provides one method for learning from failures. In *model-based meta-reasoning*, an agent is endowed with a self-model, i.e., a model of its own knowledge and reasoning. When the agent fails to accomplish a given task, the agent uses its self-model, possibly in conjunction with traces of its reasoning on the task, to assign blame for the failure(s) and modify its knowledge and reasoning accordingly. Such techniques have been used in domains ranging from game playing (B. Krulwich and Collins 1992)(Ulam, Goel, and Jones 2004), to route planning (Fox and Leake 1995) and assembly planning (Murdock and Goel 2003).

However, (Murdock and Goel 2008) showed in some cases model-based meta-reasoning can only *localize* the causes for its failures to specific portions of its task structure, but not necessarily *identify* the precise causes or the modifications needed to address them. They used reinforcement learning (RL) to complete the partial solutions generated by meta-reasoning: first, the agent used its self-model to localize the needed modifications to specific portions of its task structure, and then used Q-learning within those parts to identify the necessary modifications.

In this work, instead of using reinforcement learning to identify the modifications necessary in a task model, we evaluate the hypothesis that model-based meta-reasoning may also be used to identify the appropriate RL space for a specific task. The learning space represented by combinations of all possible modifications to an agent's reasoning and knowledge can be too large for RL to work efficiently. One way in which this complexity can be addressed is through the decomposition of the learning problem into a series of smaller sub-problems (e.g. (Dietterich 1998)). This research examines how an agent may *localize* learning within such a decomposition through the use of model-based meta-reasoning. We evaluate this hypothesis in the context of game playing in a highly complex, non-deterministic, partially-observable environment.

Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique in which an agent learns through trial and error to maximize rewards received for taking particular actions in particular states over an extended period of time (Kaelbling, Littman, and Moore 1996). Formally, given a set of environmental states $\mathcal{S}$, and a set of agent actions $\mathcal{A}$, the agent learns a policy, π, which maps the current state of the world $s \in \mathcal{S}$, to an action $a \in \mathcal{A}$, such that the sum of the reinforcement signals r are maximized over a period of time. One popular technique for learning such a policy is called Q-Learning. In Q-Learning, the agent calculates Q-Values, the expected value of taking a particular action in a particular state. The Q-Learning update rule can be formulated as $\mathcal{Q}(s,a) = \mathcal{Q}(s,a) + \alpha(r + \gamma \max_{a'} \mathcal{Q}^*(s,a') - \mathcal{Q}(s,a))$, where *r* is the reward received for taking the action, $\max_{a'} \mathcal{Q}^*(s,a')$ is the reward that would be received by taking the optimal action after that, α is a parameter to control the learning rate, and γ is a parameter to control reward discounting.

Although successful in many domains, the use of RL may be limited in others due to the so-called *curse of dimension-*

ality: the exponential growth of the state space required to represent additional state variables. In these domains, the curse of dimensionality prevents the use of RL without significant abstraction of the state space. To overcome this limitation, many have investigated the incorporation of background knowledge into the RL problem (e.g. in the form of a hierarchical task decomposition). In this context, a task can be defined as a goal that the agent must achieve (e.g. buying an apple at the store) A task decomposition, therefore, can be viewed as the set of sub-goals necessary to achieve the overall goal. By decomposing the task to be learned into several smaller tasks, the state space used by the reinforcement learner can be reduced to a more manageable level.

A number of variants of hierarchical RL exist many of which are rooted in the theory of Semi-Markov decision processes (Barto and Mahadevan 2003). Hierarchical RL techniques such as MAXQ value decomposition (Dietterich 1998) rely on domain knowledge in order to determine the hierarchy of tasks that must be accomplished by the agent, as does our approach. However, in our approach, the agent uses model-based meta-reasoning to determine the portion of the task structure over which the reward should be applied during task execution. Furthermore, many hierarchical methods focus on abstractions of temporally extended actions for the hierarchy (Sutton, Precup, and Singh 1999); our approach uses the hierarchy to delimit natural partitions in non-temporally extended tasks.

Anderson, et. al. (Anderson et al. 2006) have applied meta-reasoning in the context of RL. In their "metacognitive loop" (MCL) architecture, a metareasoning component monitors the performance of an RL-based agent. In MCL, meta-reasoning plays the role of monitoring for and correcting problems in RL. This is in contrast to the work described here where meta-reasoning is used to focus RL during normal operation. The two approaches are likely to be complementary, however.

The FreeCiv Game

The domain for our experimental investigation is a popular computer war strategy game called FreeCiv. FreeCiv is a multi-player game in which a player competes either against several software agents that come with the game or against other human players. Each player controls a civilization that becomes increasingly modern as the game progresses. As the game progresses, each player explores the world, learns more about it, and encounters other players. Each player can make alliances with other players, attack the other players, and defend their own assets from them. FreeCiv provides a highly complex domain in which the agent must operate. In the course of a game (that can take many hours to play) each player makes a large number of decisions for his civilization ranging from when and where to build cities on the playing field, to what sort of infrastructure to build within the cities and between the civilizations' cities, to how to defend the civilization.

Due the highly complex nature of the FreeCiv game, our work so far has addressed only subtasks within the game, and not the game as a whole. Due to limitations of space, in this paper we describe only one task in detail, which we call *Defend-City*. This task pertains to the defense of one of the agent's cities from enemy civilizations.

Agent Model

We built a simple agent (that we describe below) for the *Defend-City* task. The agent was then modeled based on a variant of a knowledge-based shell called REM (Murdock and Goel 2008) using a version of a knowledge representation called Task-Method-Knowledge Language (TMKL). REM agents written in TMKL are divided into tasks, methods, and knowledge. A task is a unit of computation; a task specifies *what* is done by some computation. A method is another unit of computation; a method specifies *how* some computation is done. The knowledge portion of the model describes the different concepts and relations that tasks and methods in the model can use and affect as well as logical axioms and other knowledge necessary for inference over those concepts and relations.

Table 1 describes the functional model of the *Defend-City* task as used by model-based meta-reasoning. The overall *Defend-City* task is decomposed into two sub-tasks by the *Evaluate-then-Build* method. These subtasks are the evaluation of the defense needs for a city and the building of a particular structure or unit at that city. One of the subtasks, *Evaluate-Defense-Needs*, can be further decomposed through the *Evaluate-Defense-Needs* method into two additional subtasks: a task to check internal factors in the city for defensive requirements and a task to check for factors external to the immediate vicinity of the city for defensive requirements. These subtasks are then implemented at the procedural level for execution as described below.

The *Defend-City* task is executed each turn that the agent is not building a defensive unit in a particular city in order to determine if production should be switched to a defensive unit. It is also executed each turn that a defensive unit has finished production in a particular city. The internal evaluation task utilizes knowledge concerning the current number of troops that are positioned in and around a particular city to determine if the city has an adequate number of defenders based on available information. This is implemented as a relation in the form of the evaluation of the linear expression: $allies(r) + d \geq t$ where $allies(r)$ is the number of allies within radius r, d is the number of defenders in the city and t is a threshold value. The external evaluation of a city's defenses examines the area within a specified radius around a city for nearby enemy combat units. It uses the knowledge of the number of units, their distance from the city, and the number of units currently allocated to defend the city in order to provide an evaluation of the need for additional defense. This is also implemented as a relation in the form of the linear expression $enemies(r) + e_t \leq d$ where $enemies(r)$ is the number of enemies in radius r of the city, e_t is a threshold value, and d is the number of defenders in the city. These tasks produce knowledge states in the form of defense recommendations that are then used by the task that builds the appropriate item at the city. The *Build-Defense* task uses the knowledge states generated by the evaluation subtasks, knowledge concerning the current

Table 1: TMKL Model of Defend-City Task

TMKL Model of the Defend-City Task	
Task	Defend-City
by	Evaluate-Then-Build
makes	City-Defended
Method	Evaluate-Then-Build
transitions:	
state: *s1*	Evaluate-Defense-Needs
success	s2
state: *s2*	Build-Defense
success	success
additional-result	City-Defended, Unit-Built
	Wealth-Built
Task	Evaluate-Defense-Needs
input	External/Internal-Defense-Advice
output	Build-Order
by	UseDefenseAdviceProcedure
makes	DefenseCalculated
Method	Evaluate-Defense-Needs
transitions:	
state: *s1*	Evaluate-Internal
success	s2
state: *s2*	Evaluate-External
success	success
additional-result	Citizens-Happy, Enemies-Accounted
	Allies-Accounted
Task	Evaluate-Internal
input	Defense-State-Info
output	Internal-Defense-Advice
by	InternalEvalProcedure
makes	Allies-Accounted, Citizens-Happy
Task	Evaluate-External
input	Defense-State-Info
output	External-Defense-Advice
by	ExternalEvalProcedure
makes	Enemies-Accounted
Task	Build-Defense
input	BuildOrder
by	BuildUnitWealthProcedure
makes	Unit-Built, Wealth-Built

status of the build queue, and the technology currently available to the agent to determine what should be built for a given iteration of the task. The Build Defense task will then proceed to build a defensive unit, either a warrior or a phalanx based on the technology level achieved by the agent at that particular point in the game, or wealth to indirectly keep the citizens of the city happy. The goal of the *Defend-City* task is to provide for the defense of a city for a certain number of years. The task is considered successful if the city has not been conquered by opponents by the end of this time span. If the enemy takes control of the city the task is considered a failure. In addition, if the city enters civil unrest, a state in which the city revolts because of unhappiness, the task is considered failed. Civil unrest is usually due the neglect of infrastructure in a particular city that can be partially alleviated by focusing the city on wealth/luxury production.

Experimental Setup

We compared four variations of the *Defend-City* agent to determine the effectiveness of model-based meta-reasoning in guiding RL. These were a control agent, a pure meta-reasoning agent, a pure RL agent, and a meta-reasoning-guided RL agent. The agents are described in detail below.

Each experiment was composed of 100 trials and each trial was set to run for one hundred turns at the hardest difficulty level in FreeCiv against eight opponents on the smallest game map available. These opponents were controlled by the standard AI that comes built into the FreeCiv engine. The same map was used for each trial across all agents. Tax rates in each scenario were set to 40% research, 30% income, and 30% luxury. Within this work, the city to be defended was built at the starting location. Any additional units available at the outset of the game (typically one or two) randomly explored in the vicinity of this city. If an opponent was discovered, these units did not actively engage in offensive behavior. The *Defend-City* task is considered successful if the city neither revolted nor was defeated. If the task was successful no adaptation of the agent occurred. If the agent's city is conquered or the city's citizens revolt, the *Defend-City* task is considered failed. Execution of the task is halted and adaptation appropriate to the type of agent is initiated. The metrics measured in these trials include the number of successful trials in which the city was neither defeated nor did the city revolt. In addition, the number of attacks successfully defended per game was measured under the assumption that the more successful the agent in defending the city, the more attacks it will be able to successfully defend against. The final metric measured was the number of trials run between failures of the task. This was included as a means of determining how quickly the agent was able to learn the task and is included under the assumption that an agent with longer periods between task failures indicate that the task has been learned more effectively.

Control Agent

The control agent was set to follow the initial model of the *Defend-City* task and was not provided with any means of adaptation. The initial *Defend-City* model used in all agents executes the *Evaluate-External* task only looking for enemy units one tile away from the city. The initial *Evaluate-Internal* task only looks for defending troops in the immediate vicinity of the city and if there are none, will build a single defensive unit. The control agent will not change this behavior over the lifetime of the agent.

Pure Model-Based Meta-Reasoning Agent

The second agent was provided capabilities of adaption based purely on model-based meta-reasoning. Upon failure of the *Defend-City* task, the agent used an execution trace of the last twenty executions of the task, and in conjunction with the current model, it performed failure-driven model-based adaptation. The first step is the localization of the

Table 2: State variables for RL Based Agents

Pure RL State Variables	Additional State Variables	Associated Sub-Task
$\leq$ 1 Allies in City		*Evaluate-Internal*
$\leq$ 3 Allies in City		*Evaluate-Internal*
$\leq$ 6 Allies in City		*Evaluate-Internal*
$\leq$ 1 Allies Nearby		*Evaluate-Internal*
$\leq$ 2 Allies Nearby		*Evaluate-Internal*
$\leq$ 4 Allies Nearby		*Evaluate-Internal*
$\leq$ 1 Enemies Nearby		*Evaluate-External*
$\leq$ 3 Enemies Nearby		*Evaluate-External*
$\leq$ 6 Enemies Nearby		*Evaluate-External*
	Internal Recommend	*Evaluate-Defense*
	External Recommend	*Evaluate-Defense*

Table 3: Failure types used in the *Defend-City task*

Model Location (task)	**Types of Failures**
Defend-City	*Unit-Build-Error, Wealth-Build-Error, Citizen-Unrest-Miseval, Defense-Present-Miseval, Proximity-Miseval, Threat-Level-Miseval, None*
Build-Defense	*Unit-Build-Error, Wealth-Build-Error, None*
Evaluate-Internal	*Citizen-Unrest-Miseval, Defense-Present-Miseval, None*
Evaluate-External	*Proximity-Miseval, Threat-Level-Miseval, None*

error through the use of feedback in the form of the type of failure, and the model of the failed task. Using the feedback, the model is analyzed to determine in which task the failure has occurred. For example, if the *Defend-City* task fails due to citizen revolt the algorithm would take as input: the *Defend-City* model, the traces of the last twenty executions of the task, and feedback indicating that the failure was a result of a citizen revolt in the city. The failure localization algorithm would take the model as well as the feedback as input. As a city revolt is caused by unhappy citizens, this information can be utilized to help localize where in the model the failure may have occurred. This algorithm will go through the model, looking for methods or tasks that result in knowledge states concerning the citizens' happiness. It will first locate the method *Evaluate-Defense-Need* and find that this method should result in the assertion *Citizens-Happy*. It will continue searching the sub-tasks of this method in order to find if any sub-task makes the assertion *Citizens-Happy*. If not, then the error can be localized to the *Evaluate-Defense-Need* task and all sub-tasks below it. In this case, the *Evaluate-Internal* task makes the assertion *Citizens-Happy* and the failure can be localized to that particular task. An extensive discussion on failure localization in meta-reasoning can be found in (Murdock and Goel 2008). Given the location in the model from which the failure is suspected to arise, the agent then analyzes the execution traces available to it to determine to the best of its ability what the type of error occurred in the task execution through the use of domain knowledge. For this agent, error localization occurs through the use of a failure library containing common failure conditions found within the *Defend-City* task. An example of a failure library used in this task is shown in Table 3. This failure library is typically the product of the existing task model in that the elements in the library map directly to the assertions that should result from executing a task within the model. For example, the task model indicates that the *Evaluate-Internal* sub-task should result in the assertion *Citizens-Happy*. One of the failures within the library would therefore be the case in which the citizens are not made happy.

If a failure has been determined to have occurred, it is then used to index into a library of adaptation strategies that will modify the task in the manner indicated by the library. These adaptations consist of small modifications to the subtasks in the defend city tasks, such as changing the *Evaluate-External* subtask to look for enemies slightly further away. These adaptations can be seen as a variation on fixed value production repair as described by (Murdock and Goel 2008). If multiple errors are found with this procedure, a single error is chosen stochastically so as to minimize the chance of over-adaptation of the agent.

Pure Reinforcement Learning Agent

The third agent used a pure RL strategy for adaptation implemented via Q-Learning (described in detail in section 2). The state space encoding used by this agent is a set of nine binary variables as seen in Table 2. This allows a state space of 512 distinct states. It should be noted, however, that not all states are reachable in practice. The set of actions available to the agent were: *Build Wealth*, *Build Military Unit*. The agent received a reward of -1 when the *Defend-City* task failed and a reward of 0 otherwise. In all trials alpha was kept constant at 0.8 and gamma was set to 0.9.

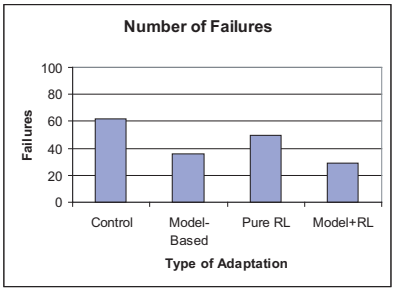

Figure 1: Number of Failures

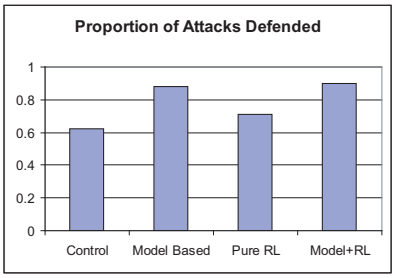

Figure 2: Proportion of Attacks Defended

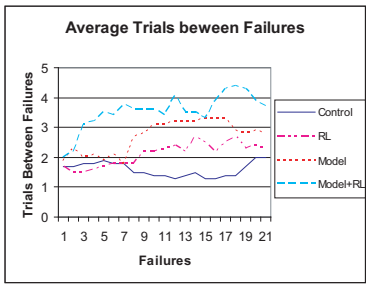

Figure 3: Average Number of Trials Between Failures

Meta-Reasoning-Guided RL Agent

The final agent utilized model-based meta-reasoning in conjunction with RL. The *Defend-City* task model was augmented with RL by partitioning the state space utilized by the pure RL agent into three distinct state spaces that are then associated with the appropriate sub-tasks of the *Defend-City* task. This essentially makes several smaller RL problems. Table 2 shows the states that are associated with each sub-task. The *Evaluate-External* task is associated with three binary state variables. Its resulting actions consist of either a recommendation that defensive units be built or a recommendation that defensive units should not be built. In a similar manner, *Evaluate-Internal* is associated with six binary state variables as shown Table 2. The actions are also a binary value representing the relation used in the pure meta-reasoning agent. There are two additional state variables in this agent that are associated with the *Evaluate-Defenses* sub-task. The state space for this particular portion of the model are the outputs of the *Evaluate-External* and *Evaluate-Internal* tasks and is hence two binary variables. The actions for this RL task is also a binary value indicating a yes or no decision on whether defensive units should be built. It should be noted that while the actions of the individual sub-tasks are different from the pure RL agent, the overall execution of the *Defend-City* task results in two possible actions for all agents, namely an order to build wealth or to build a defensive unit. Upon a failure in the task execution, the agent initiates meta-reasoning in a manner identical to the pure meta-reasoning agent. Utilizing a trace of the last twenty executions of the *Defend-City* task as well as its internal model of the *Defend-City* task, the agent localizes the failure to a particular portion of the model as described in section 5.2. If an error in the task execution is detected, instead of utilizing adaptation libraries to modify the model of the task as in the pure meta-reasoning agent, the agent applies a reward of -1 to the sub-task's reinforcement learner as indicated via meta-reasoning. The reward is used to update the Q-values of the sub-task via Q-Learning at which point the adaptation for that trial is over. If no error is found, then a reward of 0 is given to the appropriate reinforcement learner. In all trials alpha was kept constant at 0.8 and gamma was set to 0.9.

Results and Discussion

Figure 1 depicts the number of trials in which a failure occurred out of the one hundred trials run for each agent. The more successful adaptation methods should have a lower failure rate. As can be seen from the results, the meta-reasoning-guided RL agent proved most effective at learning the *Defend-City* task, with a failure rate of around half that of the control agent. The pure meta-reasoning agent with the hand designed adaptation library proved to be successful also with a failure rate slightly higher then that of the meta-reasoning-guided RL agent. The pure RL agent's performance did not match either of the other two agents in this metric. The pure RL agent's failure rate did improve over that of the control, however, indicating that some learning did take place, but not at the rate of either the pure meta-reasoning agent or the meta-reasoning-guided RL agent.

The second metric measured was the proportion of attacks successfully defended by the agent in its city. This serves as another means of determining how effectively the agent has been able to perform the *Defend-City* task. The more attacks that the agent was able to defend, the more successfully the agent had learned to perform the task. The results from this metric can be seen in Figure 2. As the default parametrization of all agents results in one defensive unit being built, this metric will not be below 50% for any agent (a city with one denfensive unit will always be able to defend against at least one attack). Both the pure meta-reasoning and meta-reasoning-guided RL agent were able to defend against around an equal number of attacks per trial (approx. 6 defenses per trial) indicating that both methods learned the task to an approximately equal degree of effectiveness. The pure RL based agent exceeded the performance of the

control (approx. 2.5 vs. 1.5 defenses per trial) but was significantly less effective then the meta-reasoning methods, once again lending support to the conclusion that the pure RL based agent is hampered by its slow convergence times. This result, coupled with the number of failures, provide significant evidence that the meta-reasoning methods learned to perform the task with a significant degree of precision. They not only reduced the number of failures when compared to the control and pure RL based agent, but were also able to defend the city from more than twice as many attacks per trial.

Figure 3 depicts the average number of trials between failures for the first twenty-five failures of each agent averaged over a five trial window for smoothing purposes. This metric provides a means of measuring the speed of convergence of each of the adaptation methods. It is assumed that as the agent learns the *Defend-City* task within the the experiment, the frequency of failures will decrease. As such, the rate in which the failures decrease can be used measure how quickly the agent learns the task. As can be seen, the meta-reasoning-guided RL agent shows the fastest convergence speed followed by the non-augmented meta-reasoning agent. The pure RL agent did not appear to improve until around the twelfth failure. After this point the control and the pure RL agent inter-trial failure rate begin to deviate slowly. Though not depicted in the figure, the performance of the pure RL based agent never exceeded a inter-trial failure rate of three even after all trials were run. This lends evidence to the hypothesis that pure RL cannot learn an appropriate solution to this problem in the allotted number of trials though the performance of this agent did outperform the control. The meta-reasoning-guided RL agent outperformed the pure meta-reasoning agent in this metric. Finally, the difference in all performance metrics for the pure RL agent and the meta-reasoning guided RL agent serve to illustrate the importance of the problem decomposition and localization in the agent's ability to learn the task.

Beyond the experiments described in this paper, we have also applied meta-reasoning-guided RL to another problem in FreeCiv. While the results for this new domain are still preliminary, this work bears mention as it helps establish the generality of the approach. In this alternative setting, an agent learns to make decisions about when to use it units to actively attack enemy units within FreeCiv. These preliminary results found that the agent, when equipped with a model of its reasoning and a decomposed RL state space appropriate for this domain, was able to significantly improve its performance (measured as the percentage of engaged enemies defeated) against the control, an agent which engages enemy units stochastically.

Conclusions

This work describes how model-based meta-reasoning may guide RL. In the experiments described, this has been shown to have two benefits. The first is a reduction in learning time as compared to an agent that learns the task via pure RL. The model-guided RL agent learned the task described, and did so faster then the pure RL based agent. In fact, the pure RL based agent did not converge to a solution that equaled that of either the pure meta-reasoning agent or the meta-reasoning-guided RL agent within the allotted number of trials. Secondly, the meta-reasoning-guided RL agent shows benefits over the pure meta-reasoning agent, matching the performance of that agent in the metrics measured in addition to learning the task in fewer trials. In addition, the augmented agent eliminates the need for an explicit adaptation library such as is used in the pure-model based agent and thus results in a significant reduction in the knowledge engineering required by the designer. This work has only looked at an agent that can play a small subset of FreeCiv. Future work will focus largely on scaling up this method to include other aspects of the game and hence larger models and larger state spaces.

Acknowledgments

This work was supported in part by an NSF (SoD) grant (#0613744) on Teleological Reasoning in Adaptive Software Design.

References

Anderson, M. L.; Oates, T.; Chong, W.; and Perlis, D. 2006. The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *J. Exp. Theor. Artif. Intell.* 18(3):387–411.

B. Krulwich, L. B., and Collins, G. 1992. Learning several lessons from one experience. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 242–247.

Barto, A. G., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4):341–379.

Dietterich, T. G. 1998. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 118–126.

Fox, S., and Leake, D. B. 1995. Using introspective reasoning to refine indexing. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. P. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Murdock, W., and Goel, A. K. 2003. Localizing planning with functional process models. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*.

Murdock, J. W., and Goel, A. K. 2008. Meta-case-based reasoning: self-improvement through self-understanding. *J. Exp. Theor. Artif. Intell.* 20(1):1–36.

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* 112:181–211.

Ulam, P.; Goel, A.; and Jones, J. 2004. Reflection in action: Model-based self-adaptation in game playing agents. In *AAAI Challanges in Game AI Workshop*.