To appear in: AIIDE-08, October 22-24, 2008, Stanford University, California.

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

Patrick Ulam, Joshua Jones & Ashok K. Goel
School of Interactive Computing, Georgia Institute of Technology, Atlanta, USA 30332
{pulam, jkj, goel }@cc.gatech.edu

Abstract

Human experience with interactive games will be enhanced if the game-playing software agents learn from their failures and do not make the same mistakes over and over again. Reinforcement learning, e.g., Q-Learning, provides one method for learning from failures. Model-based meta-reasoning that uses an agent's self-model for blame assignment provides another. In this paper, we combine the two methods. 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 effectiveness matching that of hand coded agents and with speed exceeding that of non-augmented reinforcement learning.

1 Introduction

Intelligent agents, such as software agents acting as non-player characters (NPC) in an interactive game, often fail in their tasks. However, NPC's in most commercially available interactive games generally do not learn from their mistakes. As a result, the human player quickly tires of playing the game. The human player's experience with an interactive game surely will be enhanced if the NPC's learned from their failures and not make the same mistake over and over again.

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], to route planning [Fox and Leake, 1995], to 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 it used Q-learning within the identified parts of the task structure to precisely identify the needed modifications.

In this paper, we evaluate the inverse hypothesis, viz., model-based meta-reasoning may be useful for focusing RL. 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. If, however, the agent's self-model *partitions* the learning space into much smaller subspaces and model-based meta-reasoning *localizes* the search to specific subspaces, then RL can be expedient. We evaluate this hypothesis in the context of game playing in a highly complex, extremely large, non-deterministic, partially-observable environment.

2 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 *et al.*, 1996]. Although RL is very popular and has been successful in many domains, its use is limited in some domains because of the so-called *curse of dimensionality*: the exponential growth of the state space required to represent additional state variables. In many domains, this prevents the use of RL without significant abstraction of the state space. To overcome this limitation, much research has investigated the incorporation of background knowledge, e.g. in the form of some hierarchical task decomposition, into RL. There are many variants of hierarchical RL, most 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 after task execution. Furthermore, many hierarchical methods focus on abstractions of temporally extended actions for the hierarchy [Sutton *et al.*, 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. The meta-reasoner is able to detect problems in RL, such as those that may be caused by environmental perturbations. When the meta-reasoner detects such an occurrence, measures are taken to assist the lower-level RL portion of the agent in recovering from the problems. These measures may include discarding the already learned policy, or adjusting the exploration factor in order to encourage a period of increased exploration after a perturbation is detected. Thus, in MCL, meta-reasoning plays the role of monitoring for and correcting problems in RL. In contrast, in the work described here, meta-reasoning is used to focus RL during normal operation. The two approaches are likely to be complementary.

3 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, extremely large, non-deterministic, partially-observable domain in which the agent must operate. In the course of a game (that can take a few 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.

4 Agent Model

We built a simple agent (that we describe below) for the *Defend-City* task. The agent was then modeled in 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 inferencing knowledge involving those concepts and relations. Formally, a TMKL model consists of a tuple (T, M, K) in which T is a set of tasks, M is a set of methods, and K is a knowledge base. The representation of knowledge (K) in TMKL is done using Loom, an off-the-shelf knowledge representation (KR) framework. Through the use of a formal framework such as TMKL, dependencies between the knowledge used by tasks as well as dependencies between tasks themselves can be described in such a way that an agent will be able to reason about the structure of the tasks. TMKL is similar to HTN [Erol *et al.*, 1994] but more expressive than HTN in part because TMKL enables explicit representation of sub-goals and multiple plans for achieving a goal. When Hoang, Lee-Urban and Munoz-Avila [Hoang *et al.*, 2005] designed a game-playing agent in both TMKL and HTN, they found that TMKL provides constructs for looping, conditional execution, assignment functions with return values, and other features not found in HTN. They also found that since HTN implicitly provides support for the same features, translation from TMKL to HTN is always possible. A thorough discussion of TMKL can be found in [Murdock and Goel, 2008].

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-Defend* 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* 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

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

states generated by the evaluation subtasks, knowledge concerning the current 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 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 producing wealth instead of additional troops.

5 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. This was to ensure that the *Defend-City* task would be required by the agent. The same random seed was utilized in all the trials to ensure that the same map was used. The random seed selected did not fix the outcome of the combats, however. 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.

5.1 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* 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.

5.2 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 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

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*

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, this is implemented 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. 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. This is a slight variation on fixed value production repair [Murdock and Goel, 2008], as instead of adding a special case for the failed task, the agent replaces the procedure with a slightly more general version. 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.

5.3 Pure Reinforcement Learning Agent

The third agent used a pure RL strategy for adaptation implemented via Q-Learning. 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.

5.4 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 subtask. The *Evaluate-External* task is associated with three binary state variables. Its actions are the equivalent of the knowledge state produced via the *Evaluate-External* relation in the pure meta-reasoning agent, namely a binary value indicating if the evaluation procedure recommends that defen-

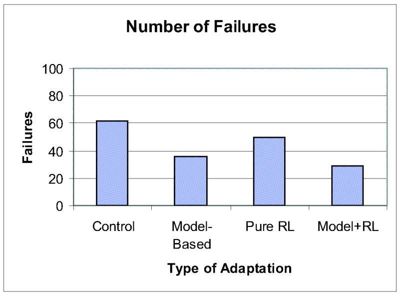

Figure 1: Number of Failures

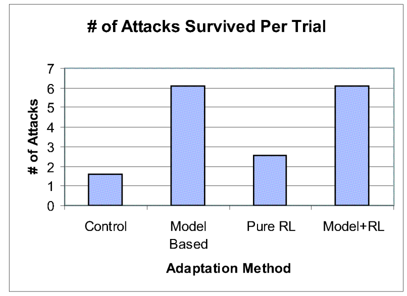

Figure 2: Average Attacks Resisted

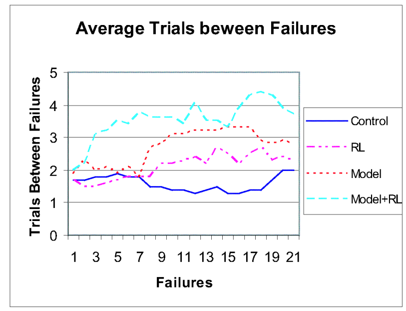

Figure 3: Average Number of Trials Between Failures

sive units 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.

6 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 success rate of around twice 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, indicating that most likely the agent had not had enough trials to successful learn the *Defend-City* task. 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 number 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. Both the pure meta-reasoning and meta-reasoning-guided RL agent were able to defend against an equal number of attacks per trial indicating that both methods learned the task to an approximately equal degree of effectiveness. The pure RL based agent performed around twice as well as the control but was less then half as effective as 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. As can be seen, the meta-reasoning-guided RL agent shows the fastest convergence speed followed by the non-augmented meta-reasoning. The pure RL did not appear to improve the task's execution until around the twelfth failed trial. After this point the control and the pure RL 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 further evidence to the hypothesis that pure RL cannot learn an appropriate solution to this problem in the allotted number of trials though it should be noted that the performance of this agent did slightly outperform that of the control, indicating that some learning did occur. The meta-reasoning-guided RL agent outperformed the pure meta-reasoning agent in this metric.

Beyond the experiment described in detail in this paper, we have also applied meta-reasoning-guided RL to another problem in FreeCiv. While our results for the new problem are still preliminary, this additional work bears some mention as it helps to establish the generality of the method. In this alternative setting, we wish to learn to make decisions about when to use offensive units to attack enemy units in FreeCiv. The agent will inspect a candidate offensive unit and an enemy unit, comparing their relative strengths. The agent then either decides to attack the enemy unit, or to hold position and defend, based on the expectation of victory or defeat. We have broken the decision making process into several distinct underlying decisions that compare various aspects of the two units in question. Reward (punishment) is provided when either of the two units dies. The reward is +1 if the enemy is defeated or -1 if the friendly unit is killed. In this setting so far we have simply compared a baseline based on random action selection to the behavior observed when we make use of meta-reasoning-guided RL. The random actor defeated the enemy unit 49% of the time, while the meta-reasoning-guided RL agent was able to kill the enemy unit 68% of the time, providing evidence that meta-reasoning-guided RL is successful in improving the performance of the agent.

7 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 converging to a solution 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 reduces the knowledge engineering burden on the designer significantly. 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.

References

[Anderson *et al.*, 2006] Michael L. Anderson, Tim Oates, Waiyian Chong, and Donald Perlis. 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, 2006.

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

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

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

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

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

[Hoang *et al.*, 2005] Hai Hoang, Stephen Lee-Urban, and Héctor Muñoz-Avila. Hierarchical plan representations for encoding strategic game ai. In *AIIDE*, pages 63–68, 2005.

[Kaelbling *et al.*, 1996] Leslie P. Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

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

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

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.