On Training Automated Agents

Jeffery A. Clouse

Department of Computer Science
University of Massachusetts
Amherst, MA 01003
CmpSci Technical Report 95-109 8

February 1, 1995

Abstract

Learning to solve problem-solving tasks is a hallmark of intelligence. Intelligent agents learn not only from their own experiences but also from the experiences of others. One would also like a computerized agent to do this: to exploit both its own experiences and those of other agents when learning to solve problem-solving tasks. To this end, we introduce a model of learner/trainer interaction that describes how a learning agent and training agent work together to help the learning agent learn. This proposed model presents the learner as an agent that must make decisions about *how* it is going to learn. For example, when should the learning agent ask the trainer for help? The training agent must also make decisions about how it interacts with the learning agent. For example, when the learning agent requests help, should the trainer provide it or ignore the request? These decisions drive the interactions, which are the mechanisms by which the training agent provides knowledge to the learning agent.

We propose to examine the issues that arise in implementing particular aspects of the learner/trainer model. We will present a restricted learner/trainer model, INTERACTIVE TRAINING, and discuss the requirements on the training agent and the learning agent. In our proposed research we will explore the trainer's ability to interact with the learner as well as the learner's ability to benefit from that interaction. Our goal is to develop a systematic method by which a training agent—human or automated—can train a learning agent effectively, allowing automated agents to be built more quickly. Furthermore, such a training method will allow agents to be built for problems that are currently deemed too difficult for automated learning agents to tackle.

Contents

1 Introduction

The ability to perform a series of actions to reach a goal is fundamental to intelligence. For example, a bear must perform a series of actions to find food, and a commuter must perform a series of actions to get to work. Both the bear and the commuter employ their reasoning abilities to perform *problem-solving tasks.*

More important than the basic ability to perform problem-solving tasks is the ability to *learn* the correct actions to choose. Having failed to find food, the bear will adapt its foraging technique, or face starvation. The bear not only learns from its own mistakes and successes, it also learns from its parents, who taught it the basics of foraging. Similarly, the commuter learns through its own experience that a particular left turn leads to an undesirably long traffic light. The commuter will also pay close attention when a co-worker drives them through a new neighborhood, taking advantage of the co-worker's knowledge of that part of town.

As intelligent agents, the bear and the commuter learn from both their own experiences and the experiences of other intelligent agents. Each is involved in a learner/trainer relationship, where the bear and the commuter are the learning agents and the bear's parents and the co-worker are the training agents. The learning agent and training agent work together to improve the learner's ability to perform the task. Sometimes the automated agent learns from the trainer's knowledge and sometimes the learner relies on itself to improve its performance. The learner/trainer relationship allows the learning agent to gain knowledge from the training agent.

The interactions that take place between the learner and trainer are the mechanisms by which the learner acquires information from the trainer. For example, the learning agent may ask the training agent for help and the training agent may either ignore the request or provide the information. Also, the training agent may decide to give the learning agent help without the learner's having asked for it. Through the interactions, the learning agent improves its ability to perform the problem-solving task.

The ability to learn in a learner/trainer relationship is a hallmark of intelligence, and raises many questions. The main question asks how the learning agent and training agent make the decisions that establish their interactions. For example, how does the learning agent decide to ask for help, and how does the training agent decide to give unsolicited help? These questions drive the proposed research within the realm of computerized learning agents

and training agents that may be either computerized or human.

Although the automated learning agent may not need to forage for food, it may need to perform other critical problem-solving tasks, such as steering an unmanned vehicle or controlling a manufacturing process. One would like the automated learning agent to learn from its own experience and also to interact with a training agent, taking advantage of the trainer's knowledge.

Other researchers have studied automated agents that learn to solve problems. One of the first was Arthur Samuel (1959, 1963), who worked with automated agents that learn from masters checkers players as well as with agents that learn from their own experience. Since Samuel's ground-breaking work, many researchers have studied the problem of automated agents that learn the correct actions to choose. These studies make assumptions about the manner in which the learning agent receives training information, requiring that particular interactions take place between the learning agent and the training agent. At one extreme are apprentice learning methods, in which the only source of training information is a training agent (Pomerleau, 1991; Sammut, Hurst, Kedzier & Michie, 1992). At the other extreme are reinforcement learning methods, in which there is no interaction at all between the learning agent and a trainer: the agent must learn on its own (Barto, Sutton & Anderson, 1983; Watkins, 1989). More recently, integrated systems that are between these extremes have been developed where the interactions, although still fixed, are more complex (Utgoff & Clouse, 1991; Lin, 1992; Clouse & Utgoff, 1992).

The performance of these integrated systems hints at the power of the learner/trainer interactions. For example, Utgoff & Clouse (1991) developed an integrated learning agent that required only one training trial to learn to perform the task under study. Learning without the learner/trainer interaction required more training: two trials when learning from only a training agent and almost five hundred trials when learning via reinforcement learning. Lin (1992) showed that allowing a learning agent to incorporate a trainer's knowledge greatly improved the speed with which the agent learned. The learning agent in Clouse & Utgoff (1992) achieved learning speedups of up to two orders of magnitude versus reinforcement learning alone. Each of these systems shows the incredible benefits of integrating training with reinforcement learning and serves as inspiration for our work.

The integrated systems take different specific approaches to the interaction between the learner and trainer. In our work we present a model of the cooperating learner/trainer pair that specifies the types of interaction decisions the learning agent and training agent can

make, thus describing the decisions the two agents make about how the learner acquires *training* information. The learning agent must decide when to ask the training agent for help and when to observe the training agent, and must determine what to do with the information it receives from the training agent. The training agent's decisions involve volunteering information to the learning agent and answering the learning agent's queries. We believe that because the learning agent and training agent interact, the learning agent receives richer training information than from either training on its own or being trained solely by the training agent, and thus can learn more effectively.

We propose to examine the issues that arise in implementing particular aspects of the learner/trainer interactions, focusing on a restricted learner/trainer model, INTERACTIVE TRAINING. We will present this restricted model and discuss the requirements on the trainer and the learner. In our proposed research we will explore the training agent's ability to interact with the learning agent as well as the learning agent's ability to benefit from that interaction. Our goal is to develop a systematic procedure by which a training agent can train an automated learning agent effectively, allowing automated agents to be built more quickly. Furthermore, such a training method will allow automated agents to be built for problems that are currently deemed too difficult for learning agents to tackle.

We further expect that exploring learner/trainer interactions will lead to greater understanding of the issues in learning to solve problems and will provide insight into the interactions that may occur between intelligent computerized agents and other agents, notably humans.

The remainder of this proposal is organized as follows. Section 2 describes the Learner/Trainer Model. Previous work in learning to solve problems is presented in Section 3, which includes more details on the integrated methods introduced above. A particular instantiation of the model, INTERACTIVE TRAINING, and the issues associated with that instantiation are presented in Section 4. Finally, the proposal concludes with a presentation of the research methodology (Section 5) and a discussion of the expected impact of the research (Section 6).

2 A Model of Learner/Trainer Interaction

We introduce a model of learner/trainer interaction, in which the learning agent and training agent work together to improve the ability of the learning agent to perform a problem-solving task. The model specifies the type of interactions that may take place between a learner and a trainer, thus describing the mechanisms by which a learner can acquire knowledge from the environment and from the trainer. The sole goal of the learning agent is to improve its performance on the task. The training agent shares this goal, providing the learning agent with information so that the learning agent can learn. However, our model takes into account that the training agent may have its own work to perform and so must balance helping the learning agent versus completing its own task. The tasks of the learning agent and training agent are in the same domain, but may deal with different aspects of that domain. For example, the learner may be learning how to navigate a robot in a fixed environment, whereas the training agent's task involves navigating around moving obstacles.

In this model, the learning agent and training agent employ two classes of actions. The first class of actions are applicable to the problem-solving task. These actions manipulate the task environment, for example, moving a robot forward, or turning the robot. The second class of actions, called meta-actions, define the interactions that occur between the learning agent and training agent. To distinguish between the mechanisms for deciding which action to choose and which meta-action to choose, we will refer to the first as simply a *domain decision-policy* and to the other as an *interaction decision-policy.* The mechanism for deciding which task action to choose is a domain decision-policy, which is a mapping from the states of the task to task actions. A set of interaction decision-policies are employed by the training agent and learning agent to determine which meta-action to choose.

The following sections present more details on the learning agent, the training agent, and the interactions between the learner and trainer.

2.1 The Learning Agent

The learning agent is a computerized agent that is attempting to learn to solve a problem solving task by developing a domain decision-policy. To be successful, the automated learning agent must construct a policy that will allow it to satisfy the goal of the problem. In learning the policy, the agent takes advantage of two sources of knowledge. The first source is the task environment, which provides feedback to the learning agent based on how well the agent

is performing. The learner can thus perform trial-and-error experiments in the task to gain *knowledge* about the task and can improve its domain decision-policy without the aid of a trainer. The second source of knowledge is, of course, the training agent, which provides the learner with task knowledge to help the learner improve its performance.

Figure 1 depicts the model of the learning agent. The figure represents a simple conditional branch (diamond), the meta-actions (circles), and the interaction decision-policies (stars).

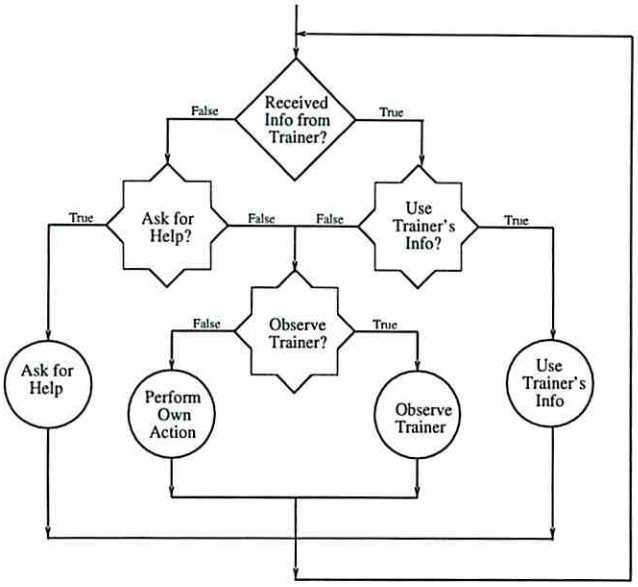


Figure 1. Learning Agent Model

The only conditional branch is at the top of the diagram and represents the fact that at each time step the learning agent determines first whether the training agent has provided it with any information regarding the current state of the task.

The learning agent makes use of three interaction decision-policies, each of which pertains to different aspects of the interaction with the training agent. If the learning agent has not received a task action from the training agent, it can decide whether to ask the training agent for help. When the learner decides to ask for help, it performs the meta-action labeled "Ask for Help" and sends a signal to the trainer notifying it of the request. The learning agent can decide to ask for help for any number of reasons, the least of which is that the agent doesn't know what to do at that point. It may also be the case that the learning agent has a good idea of what to do, but wants reassurance from the training agent. The learner may also decide not to ask for help because it has determined that the trainer's help is not

necessary.

When the learner receives training information from the trainer, it must decide the disposition of that information. The interaction decision-policy labeled "Use Training Agent's Info?" determines whether to learn from the trainer-supplied information. The learning agent may decide to use the training agent's information because the learning agent has just started learning. Or, the learning agent can decide to ignore the training agent's help; which it may do if it determines it has more knowledge about the problem than the training agent. When the learning agent learns from the training agent's knowledge, it must further decide *how* to learn: An issue that the Learner/Trainer Model raises is how the learning agent should incorporate the new information into its domain decision-policy.

The final interaction decision-policy, "Observe Training Agent?", comes into play only when the learning agent has decided not to ask for help and not to learn from the training agent's choice. This policy determines whether the learning agent should observe unobtrusively the training agent. In observing the trainer, the learner does not perform an action in its own task, but examines the trainer's task state and chosen action. The learning agent can then learn from that state-action pair. The learning agent may decide to do this, for example, when the learning agent's and training agent's tasks are similar to each other.

The final meta-action can be performed only when the learning agent has decided not to perform any of the other meta-actions. Then, the learning agent picks a task action. The learning agent is faced with the problem of choosing an action that will exploit the information it has already gleaned about the task, or performing an action to explore the task and gather more information. This decision, referred to as explore/exploit tradeoff, is an issue in reinforcement learning that will be discussed in Section 3.2.

2.2 The Training Agent

The training agent may be an automated agent or a human and may be performing a problem-solving task within the same domain as the learning agent. The training agent is already capable of functioning properly within the task domain, but may not be able to provide the learning agent with perfect information about the learning agent's task.

Figure 2 shows the model of the training agent. Like the figure of the learning agent model, this figure depicts a simple conditional branch, meta-actions, and interaction decision-policies.

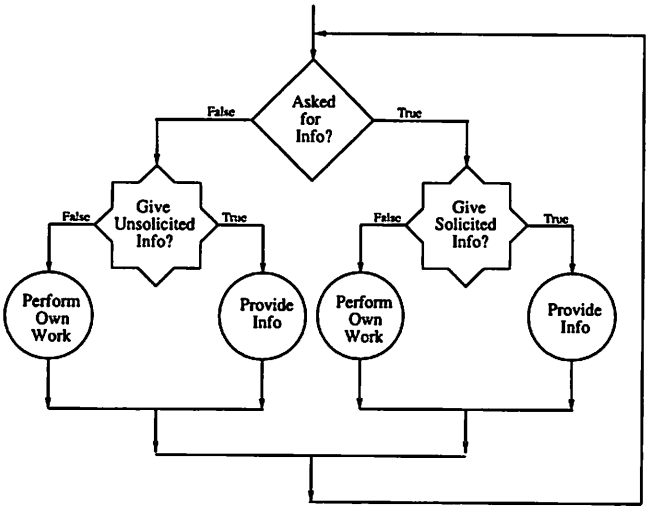


Figure 2. Training Agent Model

The conditional branch determines whether the learner has asked for help. If so, the training agent employs its interaction decision-policy labeled "Give Solicited Info?" to determine whether to supply the learning agent with the requested task information. The training agent may give the learning agent help because the learning agent has just started learning, or the training agent recognizes the learning agent's current task state as being difficult. The training agent may refuse to give help because the learning agent can benefit more from its own experience in this particular state, or maybe because the training agent can not determine what help to give.

When the training agent decides to ignore the learning agent's plea for help, the training agent performs its own work (if it has any), deciding which action to apply to its own task. If, on the other hand, the learner's request is to be granted, the trainer provides the learner with task information. It is not assumed that the proffered knowledge gives the learning agent the optimal choice to make in that situation. It may be possible that the information actually causes the learning agent to make a bad choice, but the learning agent may learn something useful by doing so. It is the case, though, that the training agent is attempting to help the learning agent learn the task and so will not mislead the learning agent maliciously.

When the learning agent does not ask for help during the last time step, the training agent decides whether to volunteer information, which it may do for the same reasons as offering solicited help. The meta-actions that can be applied by the "Give Unsolicited Info?" interaction decision-policy are the same as those under the interaction decision that

pertains to the learning agent asking for help. Although the two interaction policies might be *collapsed* into only one, there may be a need for the training agent to maintain the distinction of whether the learning agent asked for help or not. Furthermore, training agents in existing systems are either of one type, offering unsolicited help, or the other, offering solicited help. Thus, our model keeps two, separate policies.

There are many types of domain knowledge that the training agent can provide to the learning agent. The information can range in complexity from a complicated rule down to criticism of the learning agent's last action choice. The training agent can bring these different forms of knowledge to bear when training the learning agent. In our research we will examine two particular forms of task knowledge, criticism and actions (see Section 4.2.2).

3 Placing Related Work in the Model

Many different learning systems have been developed that allow an automated agent to learn to perform problem-solving tasks. Each has made assumptions about the interactions between the learning agent and training agent, thus employing particular mechanisms for allowing the learning agent to acquire task knowledge. In apprentice learning methods, in which the learning agent learns solely by observing a training agent, the learning agent always gets information from the training agent and always learns from that information. At the other extreme are reinforcement learning methods, in which there is no training agent. In these methods the learning agent learns from its own experiences and from feedback provided by the task environment. Several researchers have also suggested integrated systems that lie between these two extremes. The following sections describe these learning methods in more detail, discuss the assumptions made about the interactions between the learning agent and training agent, and show how the methods can be placed within the Learner/Trainer Model.

3.1 Apprentice Learning Methods

In *apprentice learning*, the learning agent watches as a training agent, either a human or an automated agent, performs a problem-solving task. The trainer provides the learner with the action that is appropriate to perform in the current task state. By observing the trainer-supplied actions, the learning agent develops a set of training instances where states have been paired with the appropriate actions for those states. Thus, apprentice learning is a supervised learning technique where the training instances are derived from the actions of a training agent. This type of learning has also been called "copying an existing controller" (Barto, 1990).

Like the Learner/Trainer Model, the learning agent and training agent in apprentice learning work together to train the learning agent to perform the problem-solving task. The major differences between the proposed model and apprentice learning are that the learning agent always learns from actions presented by the training agent and the training agent always provides the learning agent with those actions, whereas the interactions in the proposed model are more flexible.

Because a learning agent employing an apprentice method is learning from a training agent's decisions, it is only exposed to the narrow solution path chosen by the training agent. Researchers have found that when the learning agent is exposed to more than just a narrow path through the state-space the learning agent is better able to perform the

problem-solving task (Pomerleau, 1991; Sammut, Hurst, Kedzier & Michie, 1992).

The remainder of this section describes three systems that employ apprentice learning. The first is one of Samuel's checkers players, which uses book move information. More recent uses of apprentice learning methods include training a learning agent to drive a van (Pomerleau, 1991) and to fly a simulated aircraft (Sammut, Hurst, Kedzier & Michie, 1992). In each of these systems, the learner develops its domain decision-policy by observing a human trainer's performance on the task. This section concludes with a discussion of how apprentice learning methods fit within the Learner/Trainer Model.

3.1.1 Learning to Play Checkers using Book Moves

One of the earliest uses of apprentice learning was in training an agent to play checkers (Samuel, 1963). The trainers were master checkers players, whose move choices had been captured as they played many games and were recorded as book moves. Each game in the book-move information was recorded as the entire sequence of moves made by both players.

In learning to play the game, the learning agent simulated the re-playing of the recorded games, examining each move in turn. For a given move, the learning agent picked one of the possible move choices based on its evolving domain decision-policy. If the learning agent's choice differed from the checkers master's choice, the learning agent updated its policy accordingly. After performing the training steps for each move of each game the learning agent was able to play checkers well enough to beat moderate checkers players, but not well enough to beat master players.

3.1.2 Learning to Steer a Van

The Autonomous Land Vehicle in a Neural Network (ALVINN) project (Pomerleau, 1991) uses an apprentice learning method to train an agent to drive a modified van. By observing a human driving the van, ALVINN is able to learn to navigate on a variety of roads in different lighting and weather conditions at speeds of up to 20 miles per hour (the maximum speed of the van).

ALVINN learns how to steer the van by training a multi-layer feed-forward network (Rumelhart & McClelland, 1986). The inputs to the network are data from a video image of the scene ahead of the van. When ALVINN is being trained to avoid obstacles, laser range finder data is used in place of the video image. The outputs of the network determine the

steering direction that the van should take. When the network is being trained, the desired *direction* is given by the human driver's current steering direction, and the inputs to the network are the video image and range finder data.

ALVINN does not employ a pure apprentice learning method. Pomerleau points out that using solely an apprentice method would not allow the agent to learn to drive the van. The learning agent would not be supplied with enough variety in training instances to allow it to recover from its own mistakes, or mistakes due to the uncertainties inherent in steering a van. To remedy this situation, ALVINN takes the actual video input and steering direction and generates several more simulated training instances. These simulated instances are produced by perturbing the video input so that the van appears to be in different places relative to its current location. The steering direction for each of the simulated instances is generated based on the amount of perturbation of the image and the human's actual steering command. Thus, the learning agent is presented with the actual training instance generated directly by the action of the human driver and with several additional images.

Using this technique of training allowed ALVINN to navigate the van autonomously after less than five minutes of observing the human driver. ALVINN was able to learn to drive on two-lane, one-lane, and dirt roads under varying weather conditions. However, ALVINN's apprentice learning method was augmented by providing the learning agent with generated training instances to overcome the difficulty of training from data that does not show the learning agent how to recover from mistakes.

3.1.3 Learning to Fly a Simulated Aircraft

In a system that trains an agent to fly a simulated aircraft (Sammut, et al. 1992), human pilots flew the aircraft on a predetermined flight plan. The aircraft's state information and the pilot's actions were processed by the inductive decision-tree learning algorithm C4.5 (Quinlan, 1987; Quinlan, 1993) to produce an automated pilot. Each automated pilot was able to fly the pre-specified flight plan in approximately the same manner as the human from whom it had learned. So, an apprentice learning method was able to generate appropriate automated pilots to perform the specified task.

A key observation of this study is that the best automated pilots were trained by the worst human pilots. Because the worst pilots made mistakes and then had to recover from those mistakes, they exposed the automated learning agents to more of the task environment than just the narrow solution path to the goal. Thus, being exposed to larger parts of the

task environment near the solution path seems to allow the production of better domain *decision*-policies.

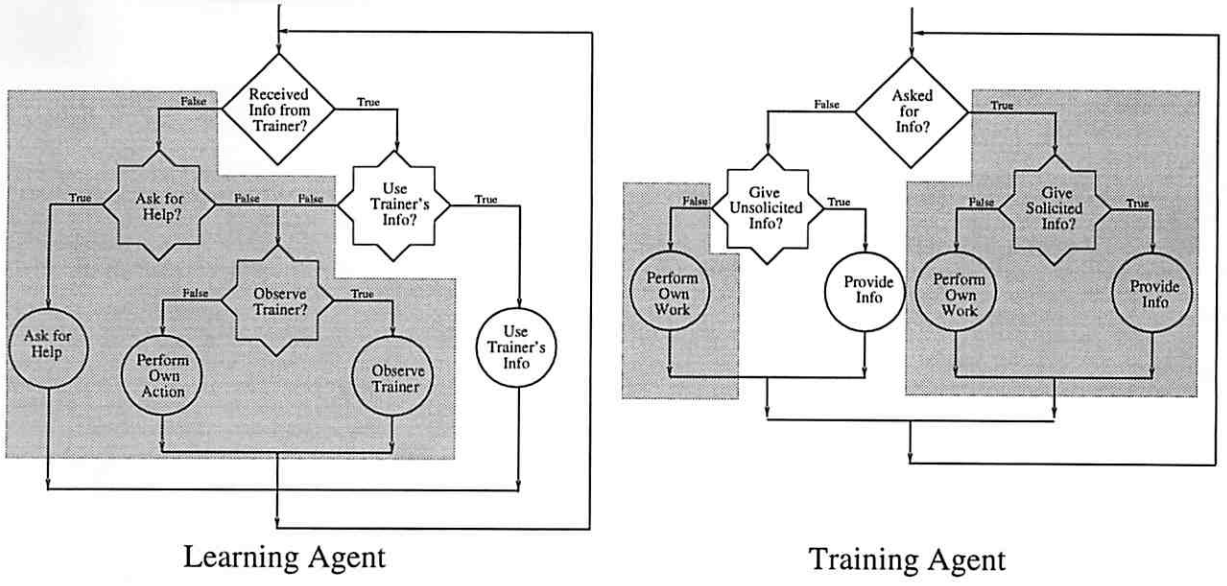
3.1.4 Discussion

In apprentice learning methods, an automated learning agent observes the actions chosen by a training agent, and learns from those actions. Unfortunately, an automated learning agent in a pure apprentice method is only exposed to the state-action selections of a training agent along a narrow solution path through the state-space, which is not enough to produce effective domain decision-policies. Two of the systems described above point to this conclusion. When the learning agent receives more information, either by generating simulated training information or having the training agent make mistakes, the learning agent is better able to produce a robust domain decision-policy, and can thus perform the task better than a learning agent that simply observes the training agent.

Figure 3 depicts a particular viewpoint of apprentice learning within the Learner/Trainer Model. The shaded areas cover the parts of the proposed model that do not apply. Note that the learning agent, depicted on the left, has only one meta-action to perform; it can train with the training agent's actions. Furthermore, the learning agent receives an action from the training agent at every time step and has no choice but to train on that action. The only applicable interaction decision-policy, "Use Training Agent's Info?", produces a fixed value of "True".

The training agent, depicted on the right of Figure 3, is never asked for help, but always provides it to the learning agent. At every time step, the training agent assesses the situation in the task and provides the learning agent with the appropriate action to take. Like the learning agent, the training agent is given only one choice of meta-action to perform.

In another interpretation of apprentice learning methods, the training agent performs its own work while the learning agent observes unobtrusively the actions of the training agent. This viewpoint also fits the proposed model, but different parts of it apply. The learning agent still has only one meta-action to perform; it can observe the training agent. The training agent, similarly, has only one meta-action; to ignore the learning agent and perform its own work. The interactions between the learning agent and training agent are as rigid in this interpretation as in the original. The main difference between the two interpretations is that in the first the learning agent is performing its own task with the help of the training agent, and in the latter the training agent is performing its own task with the learning agent



Learning Agent Training Agent

Figure 3. Apprentice Learning Learner/Trainer Model

observing. Both interpretations are valid and both point to the rigidity of the interactions in apprentice learning methods.

3.2 Reinforcement Learning Methods

In *reinforcement learning* the automated agent learns to perform its task by interacting directly with the environment (or a model of the environment), performing trial-and-error experiments and receiving feedback from an environmental critic. First, the learning agent selects an action to perform based on the current state and its evolving domain decision-policy. Then, the agent performs that action, possibly receiving a reinforcement signal from the environmental critic that informs the learning agent of how well it is performing on the task. The evaluative feedback indicates only how well the learning agent is performing, but does not give the exact action to perform. Note that a training agent is not present in this learning method.

A learning agent employing a reinforcement learning method is faced with two inherent challenges. The first is the problem of credit assignment: Which of the many actions taken by the learning agent should be credited, or blamed, for the current feedback from the environment (Minsky, 1963). The other difficulty, the explore/exploit tradeoff, arises because the agent must explore the task environment in order to learn to perform well in the environment. The learning agent must be able to determine when it has gained enough infor-

mation about the task to stop exploring and to start relying on the domain decision-policy it has learned. Much of the research in reinforcement learning is aimed at solving these two problems (Sutton, 1984; Kaelbling, 1990; Barto, Bradtke & Singh, 1991; Gullapalli, 1992; Whitehead, 1992; Thrun & Möller, 1992, and many more).

The remainder of this section describes three approaches to reinforcement learning. First, another of Samuel's checkers players will be explored. Then, two particular reinforcement learning algorithms that have been applied to many problem-solving tasks will be described. This section concludes with a discussion of the difficulties inherent to reinforcement learning methods and of how the methods can be placed within the proposed model.

3.2.1 Learning to Play Checkers via Self-Play

One of the earliest uses of reinforcement learning was in another of Samuel's checkers players (Samuel, 1959), in which the learning agent learns from its own evaluation of how well it is doing. While the learning agent is learning the problem-solving task, it develops an evaluation function that determines the value of each state. Whenever faced with a choice of possible moves to make, the learning agent chooses the one with the highest evaluation.

After the task action has been chosen, the evaluation function is changed so that the value of the current state will be more like the value of the state that results from performing the chosen action. That is, the value of the next state is backed-up to be the value of the current state. This idea of backing-up values led to the development of temporal difference learning (Sutton, 1988) which is the main technique used in other reinforcement learning methods to deal with the credit assignment problem.

Using this simple technique of backing-up state values, this checkers playing system was able to learn to play checkers well enough to be considered a moderate checkers player. Note that there was no formal exploration policy; exploration resulted from the changing evaluation function only.

3.2.2 Associative Search Element/Adaptive Critic Element

In this reinforcement learning method, the learning agent is divide into two pieces, the Associative Search Element (ASE) and the Adaptive Critic Element (ACE). The Associate Search Element/Adaptive Critic Element pair is closely related to the method of policy iteration in dynamic programming (Bellman, 1957; Barto, Bradtke & Singh, 1993), which is

a classical method employed for finding optimal policies in control engineering.

The Adaptive Search Element is the domain decision-policy, determining for each state what the correct action should be. The source of the training information for the ASE is the Adaptive Critic Element, not the environment. Each time the learning agent performs an action, the ACE provides immediate feedback to the ASE, thus helping alleviate the credit assignment problem for the ASE. The ACE acts as an evaluation function, providing a numeric value for each state of the environment, and is trained via temporal difference learning to predict the feedback from the environment. Thus, the ACE does not provide perfect information to the ASE and is still plagued by the credit assignment problem.

The mechanism used for exploration is rather simple: a small amount of uniform noise is added to the ACE, the domain decision-policy. As the policy is being developed, the small amount of noise can overpower the ASE's choice and cause another choice to be made. As the ASE is trained, though, the noise becomes insignificant and the action chosen is strictly that picked by the ASE.

The ASE/ACE pair was first applied to learning to control the cart-pole task (Barto, Sutton & Anderson, 1983). Controlling the cart-pole involves keeping a pole balanced on a movable cart and keeping the cart within the boundaries of a fixed-length track by applying equal-magnitude forces to either side of the cart. In the system built by Barto, et al. (1983), the ASE/ACE were implemented as a linear threshold unit (Nilsson, 1965). In later work, Anderson (1989) used a multi-layer feed-forward network (Rumelhart & McClelland, 1986). Both of these implementations were able to learn to control the cart-pole task after extensive training.

3.2.3 Q-Learning

Instead of maintaining both a domain decision-policy and an evaluation function, Q-learning (Watkins, 1989) combines them into a single mechanism. This reinforcement learning method is closely related to the classical dynamic programming method of value iteration, in which an evaluation function of the states is developed and the derived domain decision-policy is greedy with respect to that function.

Q-learning records, for each state-action pair, an estimate of the future environmental feedback, which is the utility of the state-action pair. Given a state and a set of actions that can be executed at that state, the action that has the highest state-action utility is the one

chosen for execution. This choice is also affected by a stochastic process that will allow the *learning* agent to explore its environment. The utility function is trained similarly to the ASE discussed above, using temporal difference learning.

Several researchers have employed Q-learning with success. Q-learning was the learning mechanism of choice in a system that learned to identify boxes and then push them around in a room (Mahadevan & Connell, 1992). Other work with Q-learning has been in the domain of simple robotics tasks and maze tasks (Lin, 1992; Singh, 1992). It has also been proved that Q-learning, under restrictive conditions, will converge in the limit to the optimal evaluation function (Watkins & Dayan, 1992).

3.2.4 Discussion

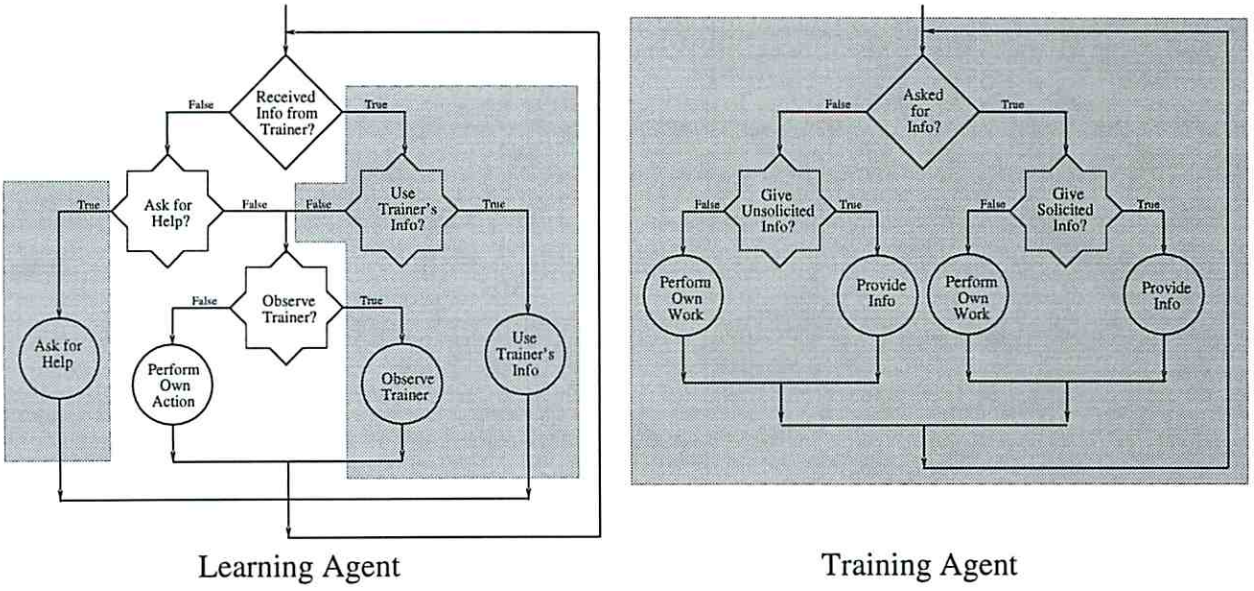
An automated learning agent in a reinforcement method has two challenges to face in learning to perform a problem-solving task. The first is the problem of credit assignment. After the learner has received a reinforcement signal from the environment it must be able to determine which of its decisions led to the receipt of the evaluative signal. The learning agent must be able to assign credit to those decisions that led to a favorable signal so they will be performed again in similar situations, and to assign blame to those that led to an unfavorable signal so they will not be chosen again. This problem is aggravated further because the reinforcement signal may be presented quite infrequently to the learning agent, after a large number of decisions have been made.

All three of the methods discussed above rely on the same type of mechanism, temporal difference learning, to address the problem of credit assignment. Temporal difference learning is the state-of-the-art, but is slow.

The other difficulty faced by a learning agent using a reinforcement learning method is the explore/exploit tradeoff. The ultimate goal of the learner is to develop a good domain decision-policy that it can use to choose operators to solve the task. Because the automated learning agent is performing trial-and-error experiments, it must be able to decide when it should explore versus when it should exploit the knowledge it already has about the task. By exploring, the learner may be able to develop a better domain decision-policy than its current one. So, the learning agent must be able to decide when to take the action that its domain decision-policy suggests, and when to choose another action that may give it more information about the task.

Samuel's checkers player does not have an exploration policy. The space that is explored is based solely on the developing domain decision-policy, and may be one reason why the system did not learn to play checkers as well as master players. The other two algorithms, ASE/ACE and Q-Learning, rely on stochastic methods to drive the exploration of the problem-solving task. Both of these exploration policies seem to work well. Other, non-stochastic exploration policies have been suggested (reviewed in Thrun & Möller (1992) and Thrun (1992)). Such exploration techniques rely on frequency counts and time stamps in discrete domains to choose infrequently or not recently executed state-action pairs. Another of these non-stochastic techniques, the *competence map* (Thrun & Möller, 1992), builds a mapping between the states of the task and a prediction of the competence of the domain decision-policy. The automated learning agent attempts to explore regions of the task environment where the competence map suggests the domain decision-policy has low competence, with the aim of improving the policy in those areas.

Even though reinforcement learning has been employed successfully on several tasks, it has been shown that because of sparse environmental feedback and the combinatorially explosive nature of problem-solving tasks, a learning agent requires time exponential in the length of the optimal solution path to find an optimal solution to the problem (Whitehead, 1991). Because the learning agent needs to explore an exponential space and also has to deal with credit assignment, reinforcement learning is slow.



Learning Agent Training Agent

Figure 4. Reinforcement Learning Learner/Trainer Model

Figure 4 shows how reinforcement learning methods map to the Learner/Trainer Model. A learning agent employing a reinforcement learning method relies solely on its own experiences to learn to perform problem-solving tasks, and is depicted on the left of the figure. Note that both applicable interaction decision-policies produce constant outputs. The learning agent can only perform its own actions, either doing the current best action or choosing an action for exploration purposes. The learning agent never receives help from a training agent because a training agent is not present. For the same reason, the learning agent cannot ask the training agent for help nor observe the training agent. A training agent is not present to guide the learning agent, hence the model of the training agent, on the right of the figure, is completely shaded out. The learner/trainer interactions do not exist in reinforcement learning methods.

3.3 Integrated Learning Methods

Apprentice learning lies at an extreme within the learner/trainer framework, where the learning agent learns solely from the actions of a training agent; and reinforcement learning lies at the other extreme, where there is no training agent and the learning agent learns solely from its own experiences. This section discusses several systems that combine the two methods and thus provide a richer set of interactions.

The integrated methods can be broken into two separate groups. In the first group, discussed in the first three sections below, the training agent provides the learning agent with domain knowledge in the form of task actions to be performed. Within this group, the first two systems employ a particular set of interactions. The remaining system of the first group uses the most sophisticated interactions of any system discussed. In the second group of integrated methods, which are discussed in Section 3.3.4, the training agent gives the learning agent more complicated forms of domain knowledge.

3.3.1 Learning via Experience Replay

Lin's work (1991, 1992) focuses on how reinforcement learning can scale up to more complicated tasks than those typically attempted. Lin (1992) discusses three approaches for improving the speed of reinforcement learning. One of those approaches, called *teaching*, combines apprentice and reinforcement learning via experience-replay, in which the learning agent's experiences are stored and then repeatedly presented to the learning algorithm as though the agent were re-experiencing them. The experiences are stored as sequences of

(STATE, ACTION, RESULTING STATE, REINFORCEMENT) quadruples and are called *lessons*. In the teaching approach, a human leads the learning agent from start states to goal states, with the learning agent recording the experiences into a lesson. The learning agent then repeatedly replays the human-generated lessons as experiences.

Although Lin's approach to combining apprentice and reinforcement learning requires the human trainer to develop complete sequences of actions that are appropriate for teaching the learning agent, the results of his experiments indicate that the learning agents were able to learn *much* more quickly with teaching. In one case, the learning agent was able to learn a task via teaching that it did not learn with reinforcement learning alone.

3.3.2 Learning via Interactive Teaching

In Clouse & Utgoff (1992), a simple interface was added to a reinforcement learning method to allow a human to interact with the automated learning agent on-line and in real-time. While the agent is learning the problem-solving task via a reinforcement method, the human monitors the learner's performance. The human injects an action to the learning agent whenever he or she desires, to provide the learning agent with a correct choice to make at that point in the task. At each time step the learning agent senses whether the human trainer has supplied such an action. If so, the learning agent uses the action as both a high quality reinforcement signal and as the current action for the task being performed. Otherwise, the learning agent makes its own action choice based on its developing domain decision-policy. In an experiment with the familiar cart-pole task, the learning agent learned the task with *two orders of magnitude less* training effort than that required by plain reinforcement learning, while receiving only an average of seven actions from the trainer. Each of the actions was a single key-stroke.

This approach to integrating apprentice and reinforcement learning showed that the development of the domain decision-policy can be *greatly* improved when a training agent actively attempts to teach the learning agent to perform the problem-solving task. However, as with Lin's system above, the trainer decided when such training information was appropriate for the learner.

3.3.3 Learning via Querying an Expert

In another integrated method, Utgoff & Clouse (1991) employed the simple reinforcement learning method used by Samuel in his checkers player: the value of the current state should

be changed to reflect the value of the next successor state chosen by the learning agent. The learning agent inferred that the state resulting from the action chosen by the training agent should have a higher value than all of the other possible states. As the learning agent was training with the reinforcement learning method, whenever the percentage difference between the value of the current state and the value of the successor state exceeded a specific threshold, the automated training agent was queried and would inform the learning agent of the appropriate value for the current state. The learning agent in this method is more sophisticated than any previously mentioned learning agent: it can decide whether or not to ask the training agent for help.

Because the learning agent could ask for help, it was able to learn to perform the task in *only one training trial*, whereas an apprentice learning method required two trials and the reinforcement method required almost five hundred. Furthermore, the frequency of training agent's actions used by the learning agent dropped sharply as the learning agent learned to perform the task.

3.3.4 Task Decomposition and Shaping

Task decomposition is a method by which a human expert provides the learning agent with a sequence of subtasks to learn, thereby simplifying the overall learning task. For example, Mahadevan & Connell (1992) describe a system in which the learning agent's task is decomposed into three domain specific subtasks. The human expertise comes in the form of the *a priori* decomposition of the task into subtasks, the priorities of the subtasks, and the identification of the subtasks to the learning agent.

Shaping, a method used in operant conditioning (Skinner, 1938), has been adopted recently as a method for training automated learning agents. The learning agent is trained on increasingly complex approximations of the required task (Gullapalli, 1992), and would not be expected to learn to perform the task without this aid. In this case, the human expert specifies the approximations that allow the learning agent to learn its task.

Task decomposition and shaping each take advantage of human expertise to lessen the difficulties of learning in problem-solving tasks. Unlike the previously discussed integrated methods, in these two methods the training agent gives the learning agent domain knowledge that is not simply a task action.

3.3.5 Discussion

An automated learning agent employing one of these integrated methods learns from both its own experiences in the task as well as the experiences of a training agent. However, the interactions are still more restricted than in the proposed Learner/Trainer Model.

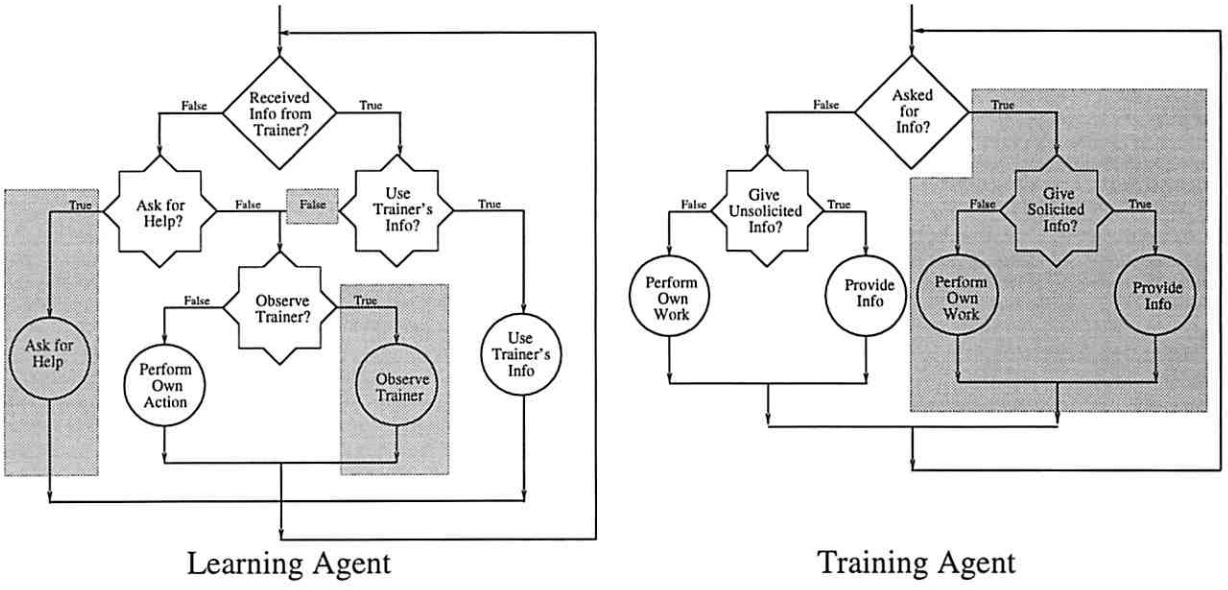


Figure 5. An Integrated Learning Learner/Trainer Model

Figure 5 depicts the learning agent and training agent for Lin's system (Section 3.3.1) and Clouse's & Utgoff's system (Section 3.3.2). The learning agent can employ two meta-actions: perform its own task action and perform the training agent's task action. The learning agent's interaction decision-policies, as in all systems discussed previously, are fixed. When the learning agent receives input from the training agent it must use it as training information. When the training agent does not provide the learning agent with help, the learning agent is on its own.

The training agents in these two systems can decide whether to give the learning agent unsolicited help or not. These training agents are not automated: the learning agent gets a human trainer's complete attention. The two systems show that allowing a training agent to provide occasional information to the learning agent will improve the learning agent's ability to learn its task.

The learning agent in Utgoff & Clouse (1991) (Section 3.3.3), is the most sophisticated learning agent discussed. The interaction decision-policy for deciding when to ask the train-

ing agent for help is an automated policy with two possible outcomes. The policy indicates *whether* to ask the training agent for help or to perform a task action.

Each of these integrated methods hints at the power of learner/trainer interactions. The results all show marked decreases in the amount of training necessary for the learning agent to learn to perform the domain task. These performance improvements serve as an inspiration for our work of studying learner/trainer interactions.

4 An Instantiation of the Model

The purpose of the proposed research is to study certain aspects of learner/trainer interaction because such interactions are the mechanisms by which the learner acquires knowledge from the trainer. In particular, the focus will be on a restricted learner/trainer model we call INTERACTIVE TRAINING, in which a trainer—human or automated—trains an automated learning agent while the learner performs the task. Our main objective is to establish a systematic method by which the training agent can train the learning agent effectively. Such a procedure will tell the training agent when to provide the learning agent with information, thus implementing the "Give Unsolicited Info?" interaction decision-policy, and will further specify the required form of that information. We will also investigate the manner in which the learning agent incorporates the training agent's proffered knowledge into its domain decision-policy.

Although we will be focusing on the restricted model INTERACTIVE TRAINING, two of the major research issues also apply to any learner/trainer model, and are thus pertinent when considering how an automated learning agent acquires information from a training agent. One of the issues in INTERACTIVE TRAINING that we will address is how the learning agent can adjust its policy to incorporate the trainer's information. This issue is relevant to *all* models that deal with learner/trainer interaction because the learner must update its policy based on information that it receives. We will also examine the issue of the form of information that the learner receives from the trainer. This issue is also applicable to any situation in which a training agent communicates with a learner.

The following sections describe the learner and trainer in INTERACTIVE TRAINING and discuss the research issues.

4.1 Interactive Training

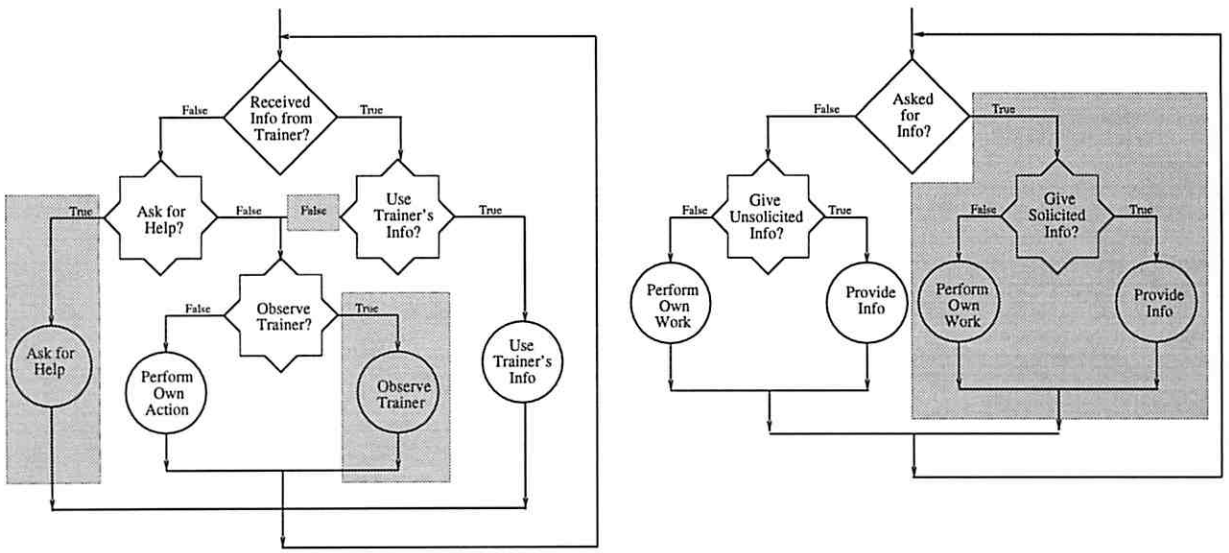
In INTERACTIVE TRAINING, a trainer provides knowledge to an automated learning agent while the learning agent is learning to perform in the domain. As will be discussed below, this model is a restricted form of the Learner/Trainer Model: both the learning agent's and the training agent's interaction decision-policies are more limited than in the general model. However, other aspects of the learning agent and training agent remain the same.

4.1.1 The Learning Agent

The learning agent in INTERACTIVE TRAINING, like the more general learning agent described in Section 2.1, attempts to develop a policy that informs the agent of which action to perform in any given domain state. Also like the more general learning agent, this agent has access to two sources of knowledge: environmental feedback and the information offered by the training agent. However, the interaction decision-policies of the learning agent are limited:

- The learning agent will always train from the training agent's information, if there is any,
- The learning agent will never choose to observe the training agent, and
- The learning agent will never ask for help.

Thus, each of the learning agent's interaction decision-policies is a constant: "Ask for Help?" is false, "Observe Training Agent?" is false, and "Use Info?" is true. A depiction of this learning agent is on the left-hand side of Figure 6. The shaded areas represent parts of the more general Learner/Trainer Model that do not apply to INTERACTIVE TRAINING.



Learning Agent Training Agent

Figure 6. INTERACTIVE TRAINING

Although these restrictions seem to constrain the learning agent to the point where learner/trainer interactions cannot be studied, an issue that remains with respect to the

learning agent is how to incorporate the training agent's task knowledge into the domain *decision*-policy. This research issue will be discussed in Section 4.2.3. Furthermore, as is discussed in the next section, the constraints on the training agent are not as great.

The description of the learning agent in the more general Learner/Trainer Model does not specify the mechanism by which the learning agent learns its domain decision-policy. The learning agent in our research will employ reinforcement learning methods (Section 3.2) for this purpose. These methods allow the learning agent to learn from environmental feedback, which is the sole task knowledge the learning agent receives when the trainer is not providing training information. Furthermore, reinforcement learning methods can adapt the learning agent's domain policy on-line, while the learning agent is performing the task.

4.1.2 The Training Agent

The training agent in INTERACTIVE TRAINING is expert at performing the domain task—or at least some part of it—and may be human or computerized. This training agent, like the one in the more general Learner/Trainer Model, is attempting to help the learning agent learn to solve the problem by providing task knowledge. However, only the interaction decision-policy "Give Unsolicited Info?" comes into play. The interaction decision-policy "Give Solicited Info?" is superfluous because the learning agent does not ask for help. This training agent is depicted on the right-hand side of Figure 6.

While the learning agent is performing the task, the training agent must have access to the state of the task in order to offer appropriate domain knowledge to the learning agent. If the training agent is a human, INTERACTIVE TRAINING requires that there exist a mechanism by which the human can watch the progress of the learning agent. This requirement necessitates the use of an interface that allows the human to acquire as much information about the state of the task and of the learning agent as is necessary to allow the human trainer to make decisions about when to help the learning agent. In our research with human trainers, the human will observe computer graphics displays of simulations of the domain tasks. These displays may be augmented with information about the learning agent, such as indicating the level of the learning agent's most recent environmental feedback.

The human trainer must also have an interface with the learning agent through which to pass training information. The interface will be simple to use so that the training agent can give the learning agent help in a timely manner. Building such interfaces is an implementation detail of the research.

4.1.3 Previous work

INTERACTIVE TRAINING is based on the work of Clouse & Utgoff (1992), which was discussed in Section 3.3.2. Unlike this earlier system, in which the training agent can only communicate task actions to the learning agent, INTERACTIVE TRAINING allows the training agent to give different forms of task knowledge. Furthermore, INTERACTIVE TRAINING relaxes the requirement that a human be the trainer, also allowing the trainer to be automated.

In addition, we will attempt to address other issues raised by this earlier effort. For example, there was little work in determining when the training agent should help the learning agent. Although Clouse & Utgoff observed that the training behavior of two different human trainers produced different outcomes, there was no understanding of why. By producing a procedure for a training agent to follow, we will better understand why certain training information is useful. This earlier research also assumed a particular method for a learning agent to learn from the training agent's knowledge; we wish to explore other methods as well.

4.2 Issues in Interactive Training

The main objective of the proposed research is to produce a methodical procedure by which a training agent can train an automated agent to perform a problem-solving task. Thus, the research has several goals. The first is to establish when the training agent should provide information to the learning agent. The second goal is to determine the form of that information. Finally, examining the manner by which the learning agent incorporates the training agent's proffered knowledge into its domain-decision policy is another goal. Collectively these goals address the issues inherent in INTERACTIVE TRAINING.

4.2.1 When to help the learning agent

In INTERACTIVE TRAINING the training agent employs the interaction decision-policy "Give Unsolicited Info?". A large part of this research is to design this interaction decision-policy in the form of a procedure that a training agent can follow. Thus, the first research question is:

Question 1 *When should the training agent provide task knowledge to the learning agent?*

We attack this problem by studying automated training agents and their "Give Unsolicited Info?" interaction decision-policies. The "Give Unsolicited Info?" policy can be a function of time, changing its output with respect to the number of task actions that the learning agent has performed. Such a policy, for example, has the training agent give help frequently at the beginning of training and less frequently as training progresses, and will be controlled by a temperature parameter that diminishes gradually. More complicated functions of time can also be employed to produce varying behaviors. For example, a policy can have the training agent help the learning agent for a period of time, then quit helping and then revert to helping after another set period of time. Cyclic policies, such as the simple policy of alternating between helping and not helping, are also functions of time.

An interaction policy need not be based on time alone; it can also be influenced by the learning agent's performance, which can be measured by the actual environmental feedback. A policy based on environmental feedback can produce one decision when the feedback is high and another when feedback is low. Because environmental feedback may be scarce, a policy based on feedback alone may be insufficient. However, an interaction policy that is based on both time and performance can be useful. An example of this type of policy is one in which a decaying average of the environmental feedback controls the decision made by the interaction policy. When negative feedback is received, for example, the training agent will decide to help the learning agent. As long as the decaying average remains below a specified threshold, the training agent will continue to help the learning agent. As soon as the environmental feedback average surpasses the threshold, the training agent will stop helping.

The interaction decision-policy can also be a function of the learning agent's current task state. For example, the training agent will know which task states are difficult or dangerous and can give the learning agent domain knowledge about those states when the learning agent enters them. This automated policy can be based on a set of heuristic rules for determining when to help.

Another issue raised by the question of when to help the learning agent deals with knowledge the training agent needs about the learning agent. Is it sufficient to observe the task states resulting from the learning agent's performance, or does the training agent also need other information about the learning agent? Such information includes the decaying feedback average discussed above. By examining automated policies that use this type of information, we will be able to determine what information about the learning agent may be needed by the training agent. This question is also asked by researchers in the field

of Intelligent Tutoring Systems (Woolf, 1988). We will also examine this related field in *develop*ing our work.

4.2.2 The form of the domain knowledge

The previous section discussed the problem of deciding when to give the learning agent task information. Another question that we will address is:

Question 2 *What form of domain knowledge should the training agent give to the learning agent?*

In our research we will examine three types of domain knowledge:

1. Individual actions,
2. Criticism, and
3. Sets of preferred or proscribed actions.

Using each of these knowledge forms raises the associated issue of how the training agent can employ the particular form to benefit the learning agent the most. The following paragraphs give more details on each of these knowledge forms.

Previous work in integrated methods have focused on providing the learning agent with individual actions (see Section 3.3). When a training agent provides an action, the learning agent receives rich knowledge: a particular state of the problem has been linked to the action that should be performed in that state. The training agent may also give the learning agent an action in order to make the learning agent explore a different part of the state space. It is not known, however, how the training agent should use actions to train the learning agent most effectively.

A training agent can also provide the learning agent with criticism of its most recent actions, much like the environmental critic does. Whitehead (1991) shows that, under certain restrictive conditions such as being able to return to arbitrary domain states, providing the learning agent with criticism after every one of its actions reduces the complexity of the learning problem to linear in the length of the solution path. Thus, a training agent's evaluative feedback, in addition to the environment's, may be beneficial to the learning agent.

Providing the learning agent with criticism will also allow the training agent to employ *the operant* conditioning technique of *shaping* (also called the method of successive approximations), in which the training agent slowly changes its presentation of reward to allow the learning agent to become progressively exposed to more difficult problems. While using shaping, the trainer will reinforce the learning agent when it performs properly a simple approximation of the requisite problem. After the learning agent is able to perform this problem, the training agent changes its use of rewards to teach the learning agent to perform a slightly more complicated problem that is still simpler than the required one. After several such approximations, the learning agent is able to perform the requisite problem.

Providing the learning agent with sets of actions represents a new approach to giving task knowledge to a learner. The set of actions can be either a set from which the learning agent should choose an action, or a set from which the learning agent should not choose an action. If the set represents actions the learning agent should choose, the training agent can present the learning agent with a limited set early in the training. As the learning agent progresses in its abilities, the training agent can relax its control over the learning agent, allowing it to choose more actions. Conversely, if the set represents actions the learning agent should not take, the set will start out large and slowly diminish in size as the learning agent learns. These two types of domain knowledge allow the training agent to keep the learning agent from making costly mistakes as it trains because the learning agent is highly constrained to choosing from a subset of the possible actions.[1]

4.2.3 Learning from the training agent's knowledge

While addressing the issues of when and how to help the learning agent, we will also ask:

Question 3 *How should the learning agent learn from the training agent's proffered knowledge?*

[1]This idea is related to 'action constraints', an issue in dynamic programming (which is further related to reinforcement learning methods—see Section 3.2). In dynamic programming, as in reinforcement learning, developing a policy requires an update operation. In some forms of dynamic programming, the update is performed for each state and *every* action that satisfies the action constraints. These constraints specify which actions are "legal" to perform in that state. To apply our idea to dynamic programming, the set of action constraints would be even further limited to those that are somehow "plausible". Note that in dynamic programming there is no interaction with a training agent, and so the action constraints are fixed.

In answering this question, the reinforcement learning methods employed to learn the *domain* decision-policy may need to be changed to take advantage of the different forms of knowledge. Even incorporating the simple domain knowledge of actions may require additions or changes to the reinforcement learning methods.

For each type of domain knowledge, there are several options regarding how to treat that knowledge. Those options are presented in Table 1. The left column of the table lists the types of knowledge that the training agent might present to the learning agent, and the right column lists the many options for each type of knowledge. Some options have further sub-options. It is a research issue to determine how to change the reinforcement learning methods to incorporate these different knowledge forms.

Type	Options
Actions	1) Perform the action and reinforce it, 2) Only perform the action, or 3) Only reinforce the action
Criticism	When it coincides with environmental feedback, either a) use it in place of the environ. feedback, or b) use it in addition to the environ. feedback
Set of preferred actions	1) Choose the best action from the set and then either a) Perform the action and reinforce it, b) Only perform the action, or c) Only reinforce the action 2) If perform one of the actions, reward it 3) If do not perform one, punish it
Set of proscribed actions	1) Choose the best NOT in the set and then either a) Perform the action and reinforce it, b) Only perform the action, or c) Only reinforce the action 2) If perform one of the actions, punish it 3) If do not perform one, reward it

Table 1. Incorporation Options for the Learning Agent

As an example, consider how the learning agent should incorporate a task action that the training agent just presented. In Clouse & Utgoff (1992), the learning agent both performs the action and reinforces it. Another possibility is to have the learning agent perform the action and then receive whatever environmental feedback results from doing so. In an alternate option, one could reinforce the training agent's action and then allow the learning agent to perform whatever action it chooses as best.

5 Research Methodology

In order to achieve our main objective of building a systematic method that a training agent can use to train an automated agent, the proposed work must address the issues raised in the previous section. The following sections discuss the experiments to be performed and the task domains in which the automated learning agent will be taught by a training agent.

5.1 Experiments

Many experiments will be performed in pursuit of answers to the questions asked in Section 4.2. The experimental variables are:

- The automated interaction decision-policy,
- The form of the training agent's knowledge, and
- The method by which the training agent's knowledge is incorporated into the domain decision-policy of the learning agent.

Because our goal is to develop a method that a training agent will use to train a learning agent, we will examine the sensitivity of this training method to changes in each of these variables. We forecast that the most sensitive variable will be the interaction decision-policies: changes in the policies will affect greatly the learning ability of the learning agent. Section 3.3.2 pointed out that giving a learning agent task knowledge can improve its performance drastically. But, we believe we can show that there exist instantiations of the "Give Unsolicited Info?" interaction decision-policy that make no difference on the performance of the learning agent in learning the task. That is, even if the training agent gives the learning agent correct information, if that information is not given in appropriate task situations it will not help the learning agent.

The statistics to measure in performing the experiments are: the *rate* at which the learning agent learns to perform the task, the *robustness* of the learning agent's learned decision-making policy and the training agent's *training count.* The rate indicates how long it takes the learning agent to develop a domain decision-policy, and the robustness indicates how good the policy is for performing the task. The training agent's training count is simply a measure of the number of times the training agent has helped the learning agent. The values of the first two statistics—rate and robustness—should be high: the learning agent should

learn quickly and learn well. We would also like to see that the learning agent requires little *help* from the training agent in order to achieve a high learning rate and a high robustness.

For every combination of settings of the variables, we will perform the experiments described below. Although this is a factorial experimental design, it will give us the most information about the variables and how they interact with each other. As controls against which to compare the experimental results, we will also run experiments on a learning agent that learns solely from environmental feedback and on another that learns solely from a training agent. Each experiment will progress in the following manner:

1. Set the experimental variables.

   Implement a training agent that will use the interaction decision-policy given and will present the learning agent with task knowledge in the form specified. Also implement a learning agent that incorporates the offered knowledge in the manner specified.

2. Measure the learning agent's learning rate and the training agent's training count.

   Commence training the learning agent under the conditions specified by the chosen variables. The training run will be terminated when the learning agent can meet the stopping criterion of solving a pre-chosen set of task problems. The number of training instances necessary to achieve this criterion will be recorded as the learning rate of the learner. The training agent's training count will also be recorded. So that we can generate learning curves that show how the performance of the agent progresses over time, we will also record the number of pre-chosen problems that the agent can solve for each time that it is tested.

3. Measure the robustness of the learned domain decision-policy.

   Employ the learning agent's learned domain policy on a set of pre-chosen domain problems and record the number of these problems that are solved. The higher the value of this metric, the more robust the domain decision-policy.

We can perform various statistical analyses on the results of the experiments that will enable us to formulate a procedure for the training agent to utilize. Two particular analyses to perform are contingency table analysis, which gives an indication of the relationship between experimental variables, and analysis of variance, which examines the different forms of variation that may occur in the experimental data. Together these two analyses can indicate which of the different variable settings have the most effect on the measured statistics

and which of the variables may be independent of the others. We will employ this information to formulate a procedure for the training agent to follow. These experiments may also suggest other changes that need to be made to the training procedure and may also suggest other experimental variable choices that should be examined.

5.2 Domains

To demonstrate the generality of the results of the research, and to aid in developing the research, a wide range of problem-solving tasks will be employed. Two dimensions along which the chosen tasks vary are: whether they involve continuous or discrete state-spaces and whether or not they are real-time tasks. The most important dimension along which to characterize the tasks is the difficulty of performing the task. A task's difficulty will be measured with respect to its value function, which indicates the utility of each domain state with regard to reaching a goal while avoiding pitfalls. A learning agent should be able to learn a simple task fairly well—both with and without the aid of the training agent. However, on more difficult tasks, the learning agent will fare badly without the training agent's help, yet should do well with training. That is, the impact of training the learning agent will be more evident on tasks that are more difficult. Thus, to show how well the training procedure improves the agent's ability to learn, we wish to employ difficult tasks. However, we also wish to employ simple tasks in order to debug the experimental design and to be able to engage in simple explanations of the system.

Anderson & Miller, which is a collection of "challenging control problems," presents many tasks that may be employed in the course of the research. One of these tasks is the airplane auto-lander, in which the learning agent must set the pitch angle of the aircraft in order to land it in windy conditions. The auto-lander is a continuous-space, real-time task, but the complexity of its value function is not known. We will attempt to gauge the difficulty of this task as we employ it in our research. Regardless of its difficulty as stated in Anderson & Miller, this task can be extended by also requiring the learning agent to control the throttle of the aircraft, making the task more difficult to learn.

We will also employ the domain of OTHELLO, which is a two-person game whose state-space contains approximately 10^{50} discrete states (Fawcett & Utgoff, 1992). OTHELLO has been studied by several researchers in artificial intelligence (Rosenbloom, 1982; Lee & Mahajan, 1988; Lee & Mahajan, 1990; Callan, 1993; Fawcett, 1993). Major differences between OTHELLO and the auto-lander task described above are the presence of an opponent and the difficulty of representing the states of the game to a domain decision-policy. Furthermore,

OTHELLO is a *difficult task* in which to learn because the value function is highly irregular, *containing* many non-linearities.

By examining tasks whose value functions differ in complexity, we will be able to show the generality and usefulness of our research. As the complexity of the tasks increase, the impact of training the learning agent will be more evident. The ideal domain for our use has a very low probability of being learned without the aid of a training agent. INTERACTIVE TRAINING applied to such a task will show the benefit of having a training agent train the automated learning agent: The learning agent will learn much more quickly and will learn a better domain decision-policy with the training agent's help. As we perform the research and explore the tasks mentioned above, we will be able to characterize the domains better with respect to the difficulty of the task and the necessity for a training agent. We may also discover other domains in which the training agent's influence is needed greatly because the task is too difficult to learn without the help of a training agent.

6 Conclusion

Learning to perform problem-solving tasks is a hallmark of intelligence. Another characteristic of an intelligent agent is the ability to benefit from learner/trainer interactions; *e.g.* to learn not only from the agent's own experience, but also from the experience of another agent. A high-level goal of this research is to understand the interactions that take place between an automated learning agent and a training agent because the interactions are the mechanism by which the learner acquires information from the trainer. Toward this end, we propose building a systematic procedure by which training agents can teach autonomous learning agents to perform problem-solving tasks.

Previous research in learning to solve problem-solving tasks includes work with apprentice learning methods, reinforcement learning methods, and integrated learning methods. The remarkable performance improvements of the integrated methods points to the conclusion that the interactions between the learning agent and training agent are important. However, the research in integrated methods has not examined explicitly the learner/trainer interactions.

The Learner/Trainer Model presented here takes the interactions into account and is motivated by the need to understand them. The model describes how the learning agent and training agent interact, specifies mechanisms for controlling those interactions, and is a broad framework within which to explore interaction issues. Our proposed research will examine a subset of those issues, focusing on a restricted learner/trainer model we call INTERACTIVE TRAINING. The study of this restricted model will help us meet our goal of increasing the understanding of learner/trainer interactions. Also, by laying out the specifications for INTERACTIVE TRAINING and by exploring the issues inherent in that model, we expect to produce a procedure that a training agent can use to train an automated learning agent to solve problem-solving tasks.

An implication of this research is that learning agents can be trained on two types of problems previously unsuited for learning. In the first type of problem, the learning agent employing only reinforcement learning may take a prohibitively long time to learn. With the help of a training agent, the learning agent will arrive at a correct domain decision-policy much more quickly. In the second type of problem-solving task, failing at the task may be catastrophic—or at least somewhat undesirable. Therefore, a learning agent should not be given free reign to perform whatever actions it desires. When the form of knowledge employed by the training agent is either individual task actions or sets of task actions, the

training agent *can provide* a fail-safe way for the learning agent to learn to perform the task. *In dang*erous situations, the training agent will provide actions that keep the learning agent from failing at the task, and the learning agent will learn to perform those actions. Thus, the training method that we will develop will allow agents to be built for problems that are currently deemed too difficult for automated learning agents to tackle.

Although this study is aimed expressly at building a specific training procedure and at more general learner/trainer interactions, we also see a far-reaching consequence of this research. We believe that the successors of our training procedure will supplant classical programming methods. Previous research with integrated learning methods seems to indicate that some tasks are programmed best by training. In performing the proposed research, we will gain further insight into the ability to teach automated agents. Our research will help change the nature of programming, from a task undertaken with traditional programming tools to a task in which an automated agent is *trained* to perform its functions. We believe that autonomous learning agents of the future will require less time and effort to be trained then will be required to program an agent directly.

In conclusion, this proposal introduces the Learner/Trainer Model, which unifies research in learning to perform problem-solving tasks. The model can provide focus for future work in this area as it raises many issues and provides a common framework for discussion among researchers. We expect that exploring learner/trainer interactions will lead to additional understanding of the difficulties involved in learning to solve problems and will provide insight into future human/computer interactions. Furthermore, a systematic method for use by a training agent in helping an automated agent learn to perform a task will be part of the successful completion of the proposed work with INTERACTIVE TRAINING. Such a methodical procedure will allow automated agents to be constructed more quickly and will allow agents to be built for problems that are currently deemed too difficult for an autonomous learning agent to tackle. Furthermore, we hypothesize that our research will help change the nature of programming: In the future, automated agents will be *taught* to perform their tasks.

Acknowledgements

I wish to thank my advisor, Paul Utgoff, for his guidance and for his comments on the contents and organization of this proposal. Discussions with Andy Barto, John Donahoe and Rod Grupen were helpful in formulating the ideas presented here. Carla Brodley, Joe McCarthy, Margie Connell, Neil Berkman, Gunnar Blix, and Cristina Baroglio provided valuable feedback during presentations of this material, for which I am grateful.

References

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine, 9*, 31-37.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics, 13*, 835-846.

Barto, A. G. (1990). Connectionist learning for control: An overview. In Miller, Sutton & Werbos (Eds.), *Neural Networks for Control.* Cambridge, MA: MIT Press.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1991). *Real-time learning and control using asynchronous dynamic programming*, (COINS Technical Report 91-57), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1993). *Learning to act using real-time dynamic programming*, (COINS Technical Report 93-02), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Bellman, R. E. (1957). *Dynamic programming.* Princeton, N.J.: Princeton University Press.

Callan, J. P. (1993). *Knowledge-based feature generation for inductive learning.* Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Clouse, J. A., & Utgoff, P. E. (1992). A teaching method for reinforcement learning. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 92-101). San Mateo, CA: Morgan Kaufmann.

Fawcett, T. E., & Utgoff, P. E. (1992). *Automatic feature generation for problem solving systems*, (COINS Technical Report 92-9), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Fawcett, Tom E. (1993). *Feature discovery for problem solving systems.* Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Gullapalli, Vijaykumar (1992). *Reinforcement learning and its application to control.* Doctoral dissertation, Computer and Information Science, University of Massachusetts, Amherst, MA.

Kaelbling, L. P. (1990). *Learning in embedded systems.* Doctoral dissertation, Stanford University.

Lee, K. F., & Mahajan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence, 36*, 1-25.

Lee, K. F., & Mahajan, S. (1990). The development of a world class Othello program. *Artificial Intelligence, 43*, 20-36.

Lin, Long-Ji (1991). Programming robots using reinforcement learning and teaching. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 781-786). Anaheim, CA: MIT Press.

Lin, Long-Ji (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning, 8*, 293-321.

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*, 311-365.

Minsky, M. (1963). Steps towards artificial intelligence. In Feigenbaum & Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Pomerleau, D. A. (1991). Rapidly adapting artificial neural networks for autonomous navigation. In Lippman, Moody & Touretzky (Eds.), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies, 27*, 221-234.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann.

Rosenbloom, P. (1982). A world-championship-level othello program. *Artificial Intelligence, 19*, 279-320.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing.* Cambridge, MA: MIT Press.

Sammut, C, Hurst, S, Kedzier, D, & Michie, D (1992). Learning to fly. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 385-393). San Mateo, CA: Morgan Kaufmann.

Samuel, A. (1959). Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development, 3*, 211-229.

Samuel, A. (1963). Some studies in machine learning using the game of Checkers. In Feigenbaum & Feldman (Eds.), *Computers and Thought.* New York: McGraw-Hill.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning, 8*, 323-339.

Skinner, B.F. (1938). *The behavior of organisms: An experimental analysis.* New York: D. Appleton Century.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning.* Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*, 9-44.

Thrun, Sebastian B., & Möller, Knut (1992). Active exploration in dynamic environments. In Moody, Hanson & Lippman (Eds.), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufman.

Thrun, Sebastion B. (1992). *Efficient exploration in reinforcement learning*, (CMU-CS-92-102), Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Utgoff, P. E., & Clouse, J. A. (1991). Two kinds of training information for evaluation function learning. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 596-600). Anaheim, CA: MIT Press.

Watkins, C.J.C.H., & Dayan, Peter (1992). Q-Learning. *Machine Learning, 8*, 279-292.

Watkins, C.J.C.H. (1989). *Learning with delayed rewards.* Doctoral dissertation, Psychology Department, Cambridge University.

Whitehead, Steven, D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 607-613). Anaheim, CA: MIT Press.

Whitehead, S. D. (1992). *Reinforcement learning for the adaptive control of perception and action.* Doctoral dissertation, Computer Science, University of Rochester, Rochester, NY.

Woolf, B. (1988). Intelligent tutoring systems: A survey. In Howard Shrobe & American Association for Artificial Intelligence (Eds.), *Exploring Artificial Intelligence.* San Mateo: Morgan Kaufman.