Computational Intelligence, Volume 21, Number 4, 2005

INCREMENTAL LEARNING OF PROCEDURAL PLANNING KNOWLEDGE IN CHALLENGING ENVIRONMENTS

DOUGLAS J. PEARSON

ThreePenny Software, Seattle, WA

JOHN E. LAIRD

Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor, Michigan, USA

Autonomous agents that learn about their environment can be divided into two broad classes. One class of existing learners, reinforcement learners, typically employ weak learning methods to directly modify an agent's execution knowledge. These systems are robust in dynamic and complex environments but generally do not support planning or the pursuit of multiple goals. In contrast, symbolic theory revision systems learn declarative planning knowledge that allows them to pursue multiple goals in large state spaces, but these approaches are generally only applicable to fully sensed, deterministic environments with no exogenous events. This research investigates the hypothesis that by limiting an agent to procedural access to symbolic planning knowledge, the agent can combine the powerful, knowledge-intensive learning performance of the theory revision systems with the robust performance in complex environments of the reinforcement learners. The system, IMPROV, uses an expressive knowledge representation so that it can learn complex actions that produce conditional or sequential effects over time. By developing learning methods that only require limited procedural access to the agent's knowledge, IMPROV's learning remains tractable as the agent's knowledge is scaled to large problems. IMPROV learns to correct operator precondition and effect knowledge in complex environments that include such properties as noise, multiple agents and time-critical tasks, and demonstrates a general learning method that can be easily strengthened through the addition of many different kinds of knowledge.

Key words: procedural knowledge, incremental learning, error detection, error recovery, planning, symbolic, operators, theory revision, machine learning.

1. INTRODUCTION

Of all the capabilities that are integral to the success of human intelligence, perhaps two of the most striking are our abilities to think and to learn. Thinking, as opposed to reacting, requires the ability to construct and manipulate internal models; in other words, the ability to plan. Learning allows us to adapt to changing environments and to incrementally improve our ability to perform tasks in a complex world. This research investigates an approach to these central issues: how to combine planning and learning into a single, generally intelligent agent that can function in complex and dynamic environments. In complex, dynamic environments an agent's knowledge about the environment (its *domain knowledge* or *domain theory*) will rarely be complete and correct. The agent cannot expect to have exhaustive knowledge to guide its behavior in all possible situations except in the simplest domains. Additionally, changes in the environment over the life of the agent can make any preprogrammed knowledge outdated and incorrect. Thus, to succeed, an autonomous agent must have the ability to learn new domain knowledge and correct errors in its existing knowledge.

Existing research on learning domain knowledge for planning and execution falls into two broad classes. Agents in the first class, such as reinforcement learners (e.g., Q-Learning (Watkins and Dayan 1992), Classifiers (Holland 1986), Backpropogation (Rumelhart, Hinton, and Williams 1986)) use weak inductive learning methods to directly modify an agent's execution knowledge. This knowledge is generally represented procedurally (e.g., in a neural net). By this we mean that the agent can execute the knowledge but is limited in its ability

Address correspondence to Douglas J. Pearson, 4649 Eastern Ave., N. Seattle, WA 98103, USA; e-mail: doug-web1@sunnyhome.org

© 2005 Blackwell Publishing, 350 Main Street, Malden, MA 02148, USA, and 9600 Garsington Road, Oxford OX4 2DQ, UK.

to reason directly about its knowledge (e.g., to realize that a particular part of the state space is not covered by the neural net). These systems are robust in dynamic and complex environments but generally do not support planning or the pursuit of multiple goals. As a result they are usually only applied to domains with small state and goal spaces. Also, they learn slowly as a result of their weak methods. In contrast, the second category consists of symbolic theory revision systems (e.g., EITHER (Ourston and Mooney 1990), EXPO (Gil 1992), OCCAM (Pazzani 1988)). These systems learn declarative planning knowledge through stronger methods that explicitly reason to identify and correct errors in the agent's domain knowledge. However, these more powerful systems are generally only applicable to simpler agents where actions are assumed to produce immediate, deterministic effects in fully sensed environments where there are no exogenous events.

This research explores learning *procedural* planning knowledge through deliberate reasoning about the correctness of an agent's knowledge. The system, IMPROV (Pearson 1996; Pearson and Laird 1999), uses an expressive knowledge representation so that it can learn complex actions that produce conditional or sequential effects over time. By developing learning methods that only require *limited procedural access* to the agent's knowledge, IMPROV's learning remains tractable as the agent's knowledge is scaled to large problems. IMPROV learns to correct operator precondition and effect knowledge in complex environments that include such properties as noise, multiple agents, irreversible actions, and time-critical tasks. Additionally, the deliberate reasoning about correctness leads to stronger, more directed learning and allows other knowledge sources (e.g., causal theories) to be smoothly integrated into the learning. In this way, IMPROV draws on the strengths of the existing classes of systems that learn domain knowledge, combining the powerful learning of theory revision systems with the robust performance in complex environments of reinforcement learners.

In addition to exploring the issues involved in building a system that learns procedural planning knowledge, this research also explores two related questions. First, what are the constraints and interactions between execution, planning, and learning in an agent-based system? Many existing systems that learn planning knowledge are not directly connected to an execution environment. Therefore, they do not address the question of when learning should occur or how training instances are generated. Often the approach that is taken is to consider each phase of execution, planning, and learning as being a distinct module. There has been little work done on how these phases constrain each other and on integrating them into a complete autonomous agent that learns on-line, while still functioning in the environment. For example, the time spent learning in a time-critical domain reduces the time available for planning and execution; but without learning, a task may be impossible if the agent's knowledge is incomplete or incorrect. One goal of this research is to better explore this interaction, outlining the constraints on learning, planning, and execution and presenting one approach to satisfying those constraints.

The second, related goal for this research is to develop a weak method for learning planning knowledge. The goal here is to transfer the learner's bias from the structure of the system to the agent's knowledge. Instead of encoding a strong learning bias within the system itself, the intention is to develop a method that can be easily guided by additional agent knowledge. This allows the agent to flexibly use a range of different kinds of knowledge, rather than being limited to knowledge in a single form. For example, IMPROV defaults to using a weak method for credit assignment, based on differences between training instances. Additional knowledge can be added (for instance, by adding a causal theory or through guidance from an instructor) to make the learning stronger and more directed.

IMPROV exists as both a theoretical system for the deliberate learning of procedural planning knowledge and as a specific implementation of this theory within a particular cognitive architecture, Soar (Laird, Newell, and Rosenbloom 1987). In presenting a theoretical

or functional description, as well as a specific implementation, the intention is to help identify the contributions to other learning systems. For example, a Soar agent's knowledge is encoded as production rules. In general, an IMPROV agent's knowledge representation must support efficient associative retrieval and while production rules are one choice, other alternatives (e.g., neural networks) would also be sufficient.

2. THE ENVIRONMENT AND PROCEDURAL KNOWLEDGE ACCESS

IMPROV is designed as a method for learning planning knowledge for autonomous agents. The learning is constrained by the environments that we expect the agents to face. Each property of the environment shown in Figure 1 constrains the design of the agent and IMPROV's learning method. The agent plans (rather than just relying on an execution policy that covers all states) because the state and goal spaces may be large and because planning knowledge is usually more general and can be used for many different tasks. The agent is assumed to start with some initial, approximate domain knowledge that IMPROV learns to extend and correct as the agent completes tasks in the external world.

As actions may produce complex effects, including sequential effects that occur over time, conditional effects or iterative behavior, the agent requires an expressive knowledge representation for the effects of actions. In making the representation expressive, there is a danger that the agent will become inefficient.

In many systems, the time to correct existing knowledge grows in proportion to the size of the agent's planning knowledge. This is undesirable as the agent's performance slows as it learns more. One approach to ensuring that performance does not degrade as the agent learns, is to limit the agent's access to its knowledge. IMPROV's methods only require procedural access to the agent's knowledge; that is, the planning knowledge can be executed but cannot be directly searched, examined, or modified. As the agent is unable to search or otherwise directly examine its knowledge, learning time is guaranteed to be independent of the size of the knowledge base. This allows IMPROV to use a highly expressive representation for

Environmental Property	Constraint on Agent
E1. Large, Dynamic State Space	Need to plan.
E2. Large Goal Space	Need to plan, explicit goal representation
E3. Actions with Complex Structure	Expressive Knowledge Representation
E4. Actions that Apply in a Large Range of States	Expressive, Efficient and Scalable KR.
E5. Irreversible Environmental Changes	Learning Spans Multiple Episodes
E6. Time-critical Tasks	Efficient Learning, Planning & Execution.
E7. Long-term Tasks and Continual Existence	Incremental Learning.
E8. Limited Sensing	Tolerance for Noise/Delays etc.
E9. Environmental Changes Independent of the Agent	Difficult to Predict Environment
E10. Evolving Environmental Processes	Tolerance to Changing Target Knowledge.
E11. Many Forms of Feedback.	Use of Multiple Knowledge Sources.

FIGURE 1. Environmental constraints on planning and learning.

Action: Move-Arm(q_1, q_2)
Preconditions: if ($x^2+y^2 < \cos(I^2-J^2) + \sin(I^2-J^2)$) then move-arm
Effects: $q_1 = \text{atan}(Y/X)$
$q_2 = \text{acos}((I^2-J^2+B^2)/(2*I*B))$

FIGURE 2. Complex planning knowledge.

its knowledge, because learning will not examine or otherwise analyze this representation. The agent's knowledge is divided into operators, with preconditions and actions expressed as sets of production rules. IMPROV uses a compact, intentional representation based on production rules, to keep the space requirements of the agent's knowledge as low as possible. More extensional representations, such as listing states (as in some reinforcement learning) or using sets of attribute-value pairs to define regions of state space (popular in symbolic learning) would require large amounts of space to represent complex functions, such as for the inverse kinematics operator shown in Figure 2.

3. RELATED WORK

To place IMPROV in relation to existing learners we will describe three dimensions, use them to classify learning systems and demonstrate that there is a strong correspondence between these dimensions and the ability of the learner to function in the environments outlined in Figure 1. The three dimensions are:

1. *Time to access data encoded in the representation during learning*: This is the cost of modifying the agent's domain knowledge during learning. It typically varies from

   $$>= O(\text{size of representation}) \text{ to } O(1).$$

   The intuitive distinction is whether the agent can search or otherwise manipulate the entire representation during learning. If so, we will refer to it as having *declarative* access to its knowledge. Agents with only limited access to their knowledge, such as *procedural* access where the agent can only execute the knowledge, will not have learning costs proportional to the size of the representation. For example, a theory revision system that reasons directly over the conditions in a STRIPS operator (Fikes and Nilsson 1971) to determine if a condition could be removed or generalized, will generally take time proportional to the number of conditions so that we could classify this as having declarative access to the representation.
2. *Size of the representation*: This is the amount of space required to describe the agent's knowledge about the domain. Let us assume the state space can be divided into *n* regions for when a particular action should be chosen. In representing these regions, the size of the representation in learning systems typically varies from

   $$O(\text{exponential}(n)) \text{ to } O(n) \text{ to } O(\log(n)).$$

   We will use the term *extensional* to refer to representations that list each state when an action can be taken. These representations are proportional to the size of the state space and so grow exponentially as the number of regions and the size of the space grows

(e.g., methods based on propositional logics or simple tables of values). In the middle of the range are *semi-extensional* representations, typically based on a disjunctive normal form. These representations use a series of attribute-value conditions to enumerate each region, but are still proportional to the number of regions. At the smallest end are fully *intentional* representations that use an unrestricted function to represent when an action should be selected, potentially leading to representations that are sub-linear in their space requirements such as the example shown in Figure 2. That is to say they require fewer terms in the function than there are regions in the state space. In contrast, if all of the values for x, y, i, and j that satisfied the equations were enumerated by region, or worse by value, it should be clear that this might require a lot of space.

3. *Explicit reasoning about the agent's knowledge*: This dimension defines the degree to which the agent reasons about the correctness of its knowledge, rather than just its task performance. This is a form of meta-level reasoning, as the agent reasons about the underlying knowledge that leads to task performance, rather than just reasoning based on task performance. We will use *deliberate* learner to refer to an agent that reasons about errors in its knowledge and *implicit* learner to refer to agents that limit their reasoning to task performance.

Existing approaches generally fall into the categories summarized in Figure 3. Learning costs increase up and to the right as the agent's knowledge is scaled to larger problems. We feel IMPROV represents an interesting part of the space to explore, using procedural access to a compact, intentional representation while still supporting deliberate learning.

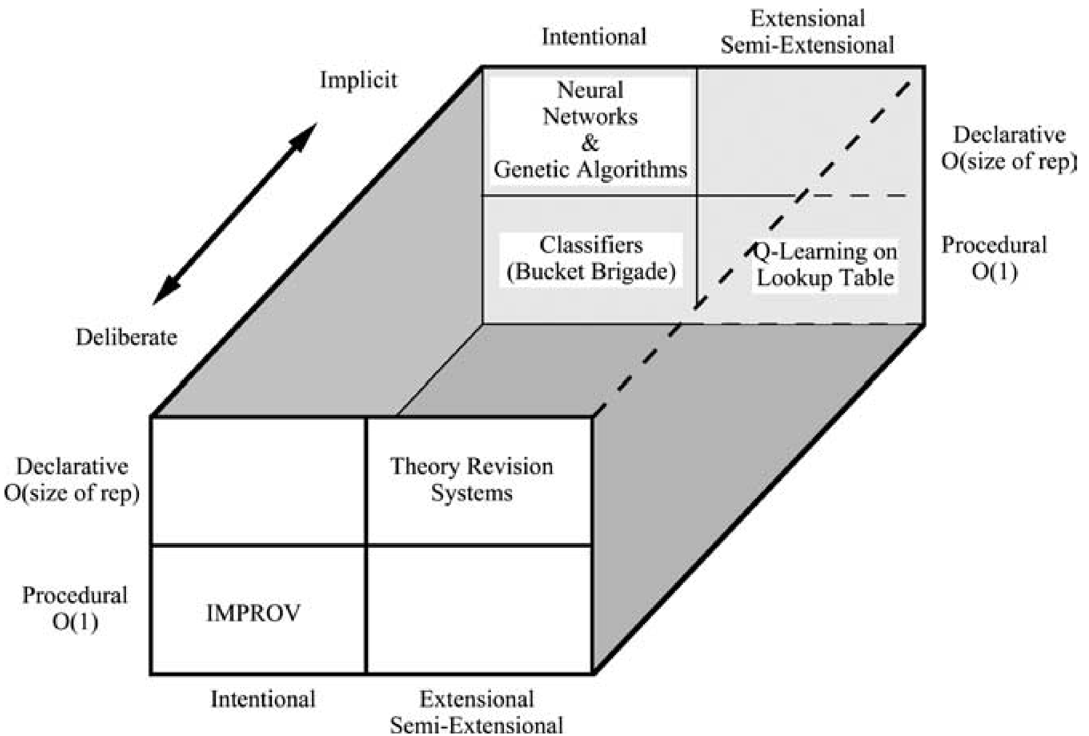


FIGURE 3. Classification of learning systems.

3.1. Deliberate Learning of Declarative Representations

This section describes some representative examples of systems that learn by deliberately reasoning about errors in the agent's knowledge. These systems typically rely on full, declarative access to the agent's domain knowledge and typically represent that knowledge in a semi-extensional form as a list of symbolic conditions for when to take an action. They are not generally applicable to the more challenging environments of Figure 1. EXPO (Gil 1993, 1994), LIVE (Shen and Simon 1989) and OBSERVER (Wang 1995, 1996) share a similar STRIPS-like representation. EXPO learns by designing experiments to refine initially overgeneral preconditions. Errors are detected by explicit monitoring of operator preconditions and actions. LIVE and OBSERVER learn by executing actions in the environment and observing the changes in the state and rely on the assumption that changes in the environment are due to deterministic actions of the agent. The STRIPS-like planning representation allows these systems to reason in large state and goal space and deliberately use the planning knowledge to guide future learning. However, the representation is not expressive enough to model complex actions (such as those that produce sequential effects) and learning time is proportional to the size of the operator structures. Sensing is also assumed to be complete and noise-free and changes in the environment are assumed to be due only to the agent's actions. EITHER (Ourston and Mooney 1990), NEITHER (Baffes and Mooney 1993), FOIL (Quinlan 1990), and FOCL (Pazzani, Brunck, and Silverstein 1991) learn Horn-clause propositional logic and have similar strengths and weaknesses as the STRIPS-based methods.

3.2. Implicit Learning of Intentional Representations

Learning is implicit when incorrect task performance is the focus, rather than incorrect knowledge. This may prevent the learner from localizing the correction. For example, an agent that drives through a corner too quickly may skid. The incorrect performance (the skid) leads to negative feedback and credit assignment to each step in driving through the corner, including turning, braking etc. Eventually the agent may drive more slowly, but this will never have been explicitly located as the cause of failure.

Classifiers (Holland 1986; Booker, Goldberg, and Holland 1989) represent domain knowledge as rules that can be combined into chains to produce action in the world and are therefore intentional. Credit assignment is through backward-chaining in the bucket brigade algorithm, reducing the strength of rules that do not lead to success. The genetic algorithm used to create new rules requires declarative access to the rule base. By fixing the size of that rule base performance remains constant over time, although this can lead to other problems with overtraining or task interference as previously valuable rules are discarded to make room for currently useful rules. Neural networks trained using backpropogation (Rumelhart et al. 1986) or related temporal difference methods (Samuel 1959; Sutton 1988; Tesauro 1992) similarly assign credit to all features present during a failure. Thus incorrect knowledge is only implicitly detected and removed over many instances. These systems make few assumptions about their environment and do not model the effects of the agent's actions in the world. This makes them applicable to domains with limited sensing and exogenous events and processes that change over time. The fixed size of the representation and the incremental learning algorithms mean learning remains constant during the life of the agent. On the downside, the lack of action models limits their ability to plan and they are generally only applicable to static state-spaces due to the fixed nature of their representations. Also, their implicit learning methods generally require more training instances to reach a given level of performance than stronger, more deliberate learners.

4. FRAMEWORK FOR ERROR CORRECTION

The main stages in correcting an agent's knowledge are:

1. *Classification of errors*: Determining the errors that can occur in an agent's knowledge and the range of performance failures the knowledge errors can cause.
2. *Detecting performance failures*: Recognizing that a performance failure has occurred, either during planning or during plan execution.
3. *Solving the current problem*: Deciding what is the correct course of action in the current situation. This stage is often folded into the learning phase, where the approach is to learn and then replan.
4. *Learning a general correction for the future*: Generalizing from the current situation to correct the agent's knowledge and avoid the error in the future. This learning can be further divided into three separate problems:
   a. *Credit assignment—Which operators are incorrect?* Determining which operator, or operators, had the incorrect knowledge that led to the performance failure.
   b. *Credit assignment—How are the operators incorrect?* Having identified the incorrect operator, or operators, the agent must decide how the knowledge about that operator is incorrect. For example, which additional tests to add, to specialize the operator preconditions.
   c. *Changing the domain knowledge*: Finally, the agent must modify its knowledge to avoid the error in future. IMPROV's restriction to only executing its knowledge means that it must solve this problem without directly modifying the existing, incorrect knowledge. This is achieved by learning new knowledge that works together with the existing knowledge to correct the agent's decisions and generate the desired behavior.

5. CORRECTING OPERATOR PRECONDITIONS

5.1. Knowledge Representation

IMPROV represents the agent's domain knowledge as a hierarchy of operators. This hierarchy represents the goal-subgoal structure that is common to many symbolic reasoning systems. IMPROV's hierarchy is similar to standard top-down structured programming, with each layer of operators being analogous to a layer of procedures. The distinction is that in IMPROV, the hierarchy is built dynamically using operator precondition rules to determine which operators are included in a particular hierarchy. Control knowledge is folded into the preconditions for an operator, so that for a given goal and state, only a single operator should have its preconditions matched. High-level, abstract operators are used to represent goals that are achieved through plans. These high-level operators are implemented by a series of operators that can themselves be subplans or traditional, motor-level operators that generate external behavior. An example taken from a driving domain is shown in Figure 4(a) where the task is to correctly drive a car through a busy intersection by braking, accelerating, and changing gears at the appropriate times.

In this example, the `Set-Speed 20` operator cannot be achieved directly; thus it becomes a goal for the agent with an implementation strategy (or plan) to achieve it by braking and changing gear. During plan execution, the `Brake` and `Shift-Down` operators generate external behavior in the environment (Figure 4(a)). During planning, the same operators are used and motor-level operators, such as the `Brake` operator, are further expanded into a model of their expected effects, represented as a series of primitive, *single-effect* operators

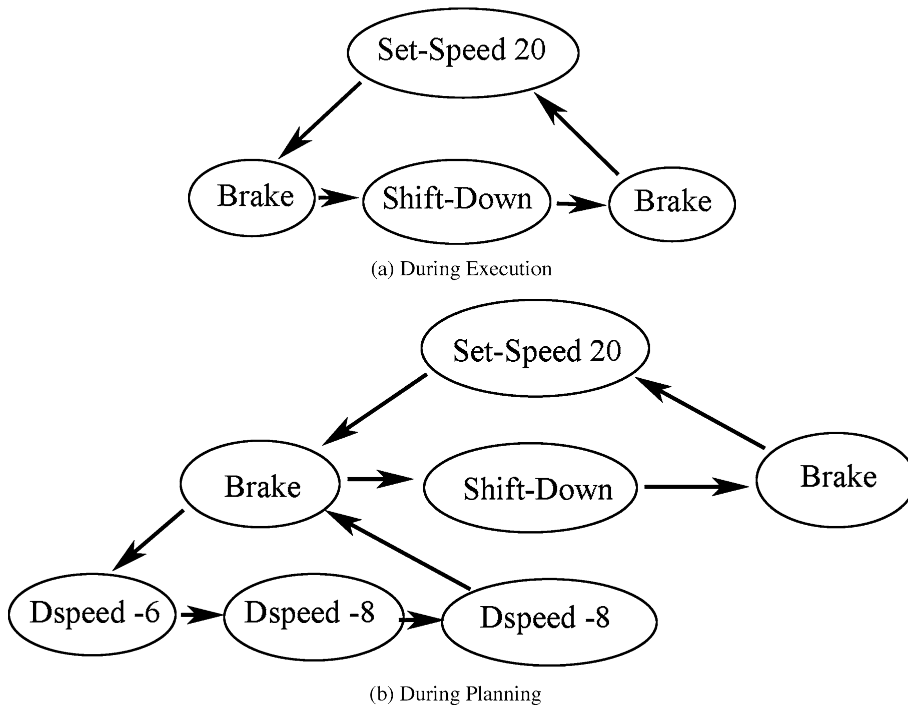


FIGURE 4. Domain knowledge as an operator hierarchy.

(Figure 4(b)). In this example, the `DSpeed` operators indicate that the rate of deceleration will change as the agent brakes. These single-effect operators are not required during plan execution, just during planning. Thus, operator actions consist of both execution knowledge that generates external behavior and planning knowledge for internal simulations. Each operator has its own preconditions, represented as a set of production rules for when to choose that operator. The hierarchies each show a single expansion for a particular problem, in this case when the car is initially traveling faster than 20 mph. If the car was initially traveling at only 10 mph, then the hierarchy implementing `Set-Speed 20` would include operators for pressing the accelerator, as the preconditions for `Accelerate` would match instead.

5.2. Classification of Errors

IMPROV's operator-based model for representing domain knowledge defines the scope of possible knowledge errors. These are limited to

1. Incorrect operator preconditions: These can be overgeneral, overspecific, or a combination.
2. Incorrect operator effects: The agent's model for the effects of actions can include extra effects, missing effects, or a combination.
3. Completely missing operators.

These errors are in the agent's knowledge about the processes of the domain. IMPROV does not correct errors in the agent's sensed data about the domain; that is, its state knowledge or state representation.

IMPROV learns new domain knowledge and corrects errors in operator preconditions (overgeneral, overspecific, or a combination) or operator effects (extra effects, missing effects, or a combination) but it cannot learn completely new operators. In certain constrained environments and with a sufficiently expressive state representation, IMPROV is guaranteed to converge to the correct operator knowledge (Pearson 1996). Incorrect knowledge can lead to a range of performance failures, either during planning or during execution. By defining the classes of knowledge-level errors we can derive the performance failures.

5.3. Detecting Performance Failures

To detect planning failures the agent requires extra knowledge about the planning process, beyond the task knowledge required to solve a problem. For instance, if the agent is unable to build a plan to solve a particular problem, it may just be that the problem cannot be solved, rather than the agent's planning knowledge is incorrect. Failures during execution can be detected by comparing the agent's planned behavior to the agent's actual behavior in the environment during execution. Traditionally, this comparison is made by explicitly monitoring each step of plan execution. The agent verifies that all of the preconditions for the next operator in the plan are satisfied before execution and then verifies that all of the expected effects have occurred. This form of explicit monitoring detects failures on the basis of the agent's ability to predict changes in the environment. In stochastic environments, or environments with multiple interacting agents or other external processes, accurate predictions may be impossible. IMPROV takes an alternative, weaker approach to detecting execution failures, by determining when the agent is unable to make progress toward its current goal. The agent may incorrectly predict environmental changes but as long as those incorrect predictions do not prevent the agent from achieving its goals, no failure is detected.

IMPROV detects a failure during plan execution if the agent reaches a state where it has conflicting suggestions (or no suggestions) on which operator to select next. This approach meshes well with IMPROV's distributed plan representation, where the plan is represented as a series of production rules that reactively guide the choice of operators during execution, rather than as a single, monolithic plan structure. Figure 5 shows an example of a plan to cross an intersection (represented as a set of rules). During execution (rules that fire are shown with shading) the agent is unaware that it must change to a lower gear, so the car stalls after `Set-Speed 0`. As a precondition for `Set-Speed 30` is that the engine is running, no operator's preconditions are matched and this is detected as an error. This approach can be extended to detecting errors in the expected effects of operator actions, by representing the effects of an operator as a sequence of more primitive operators and detecting an inability to select an operator at the lower level (see Figure 4(b)). IMPROV's method for determining when an agent is still on the path to the goal is augmented by a loop detection method based on recognizing when it returns to an earlier, similar state, to ensure that it is making progress to that goal. This recognition is based on the features of the environment that were relevant to the initial problem solving, so other irrelevant changes in the environment are ignored in determining that the agent is stuck in a loop (Pearson 1996).

IMPROV's error detection method, as with the rest of its learning, is designed as a weak method. The methods for detecting errors are general-purpose methods that make few assumptions about the environment or the agent's knowledge. It is important to realize that these weak methods can easily be made stronger by the addition of domain-specific knowledge. For example, an explicit theory of failure states can be added to IMPROV to enhance its base error detection method. Knowledge can be added easily as all of the agent's reasoning is represented as production rules within a general-purpose architecture for intelligent reasoning.

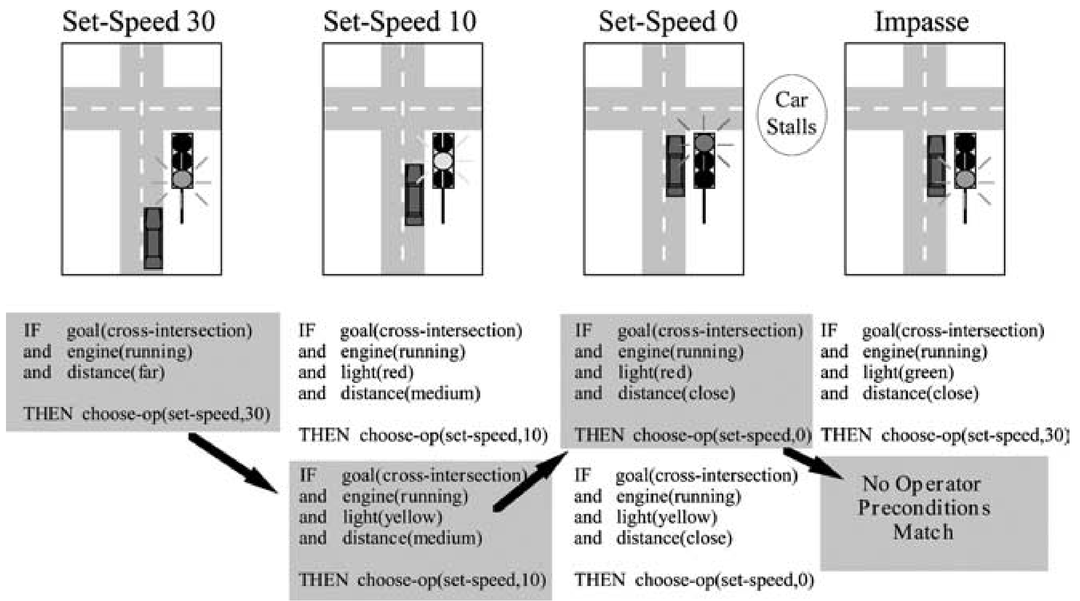


FIGURE 5. Inability to select an operator during execution signals an error.

5.4. Solving the Current Problem

IMPROV casts the task of correcting an agent's knowledge as two search problems. The first is a search through the space of plans and the second is a search through the space of operator preconditions. During the first search, plans are generated and executed in turn until the agent finds a plan that succeeds for the current goal. The sequences of operators executed during the first search are used to train an inductive learner, leading to the second search for correct operator preconditions (see Figure 6).

IMPROV builds plans through a state-space search technique called *Uncertainty Bounded Iterative Deepening* (UBID) (Pearson 1996). This is similar to iterative deepening, but the depth of the search is limited by an uncertainty measure, with each operator being assigned an uncertainty that reflects how closely its preconditions match the current state. This leads to a deeper search in areas of the search space that have earlier proved useful to the agent. Plans are represented procedurally, as a collection of rules to reactively guide the agent at each state during plan execution (see Figure 5).

IMPROV searches for alternative plans, P_i, in decreasing order of similarity to the original, incorrect plan. As each plan is generated, IMPROV temporarily assumes that the agent's planning knowledge about the effects of the operators in the plan is correct. It simulates the sequence of operators to determine the outcome of the plan, allowing IMPROV to reject plans that it believes lead to failure. Each plan that reaches the goal in the internal simulation is executed in the world, until the agent finds a plan that succeeds. If a correct plan cannot be found, IMPROV proceeds to try to find a correction within the actions of the operators.

The agent's actions may be irreversible (e.g., moving a chess piece during a game). In this case, the search for correct behavior will be spread across multiple problem-solving episodes (multiple games of chess). IMPROV recalls previous failures and previous attempted corrections when it returns to a context similar to where the original failure occurred (Pearson 1996). This allows it to search across multiple, temporally disjointed, problems. As each plan is executed, IMPROV records the sequence of states (S_{ij}), operators (O_{ij}) and the outcome

```
Main Control Loop

do
  O      := Compute-Proposal(G,S)
  result := Execute(O)

  if (result = error-detected) then
    Make-Correction(G,S)
  endif

while (result != reached-goal)
```

```
Procedure Make-Correction(G,S)
begin
  /* Search for the correct plan */
  while (current-state != G)
    P_i := UBID(G, current-state)
    do
      O_ij   := Next-step-in-plan(P_i)
      S_ij   := current-state
      result := Execute(O_ij)
      if (result = error-detected) then R_i := error-detected
      if (result = reached-goal) then { R_i:= reaached-goal ; success = i }
    while (result != error-detected and result != reached-goal)

    /* Identify key differences between correct and incorrect plans */
    /* Train inductive learner */
    foreach S_j in P_success
      important-features = Differences(S_1j,S_2j,S_3j,...,S_success,j)
      for i := 1 to success
        if (R_i = error-detected) then
          Train-SCA2-negative-instance(G, S_ij,O_ij,important-features)
        if (R_i = reached-goal) then
          Train-SCA2-positive-instance(G, S_ij,O_ij,important-features)

    /* Correct the operator precondition knowledge */
    foreach S_j in P_success
      O-current := Compute-Proposal(G, S_j)
      O-new     := Test-SCA2(G, S_j)
      if (O-new != O-current) then
        Learn-Reject-Operator(G, S_j, O-current)
        Learn-Propose-Operator(G, S_j, O-new)
end
```

FIGURE 6. Summary of the correction method.

of the plan (success or failure recorded in R_i). IMPROV uses these records as positive and negative training instances for the inductive learner, once a successful plan has been found.

5.5. Learning a General Correction for the Future

IMPROV delays learning until a successful plan has been discovered, which allows it to use the comparisons between the successful plan and the unsuccessful attempts to improve credit assignment during learning. IMPROV's delayed learning also helps it avoid incorrect early learning. This can be particularly harmful to an active learner (one that is learning while it performs tasks in the environment) because early learning influences the later instances the agent will see. For example, a driver that learns to stop for every bus, rather than just school buses, will avoid passing buses and thus may take a long time to discover that its original learning was overgeneral.

1. *Credit assignment—Which operators are incorrect?* IMPROV's approach to determining the operators that caused a failure is to compare the successful plan to the original, incorrect plan, and use the differences to determine the operators that are in error. For each state in the successful plan, IMPROV determines which operator would have been chosen using the original, possibly incorrect, planning knowledge applied to that state. This original operator is compared to the operator used in the successful plan. Figure 7

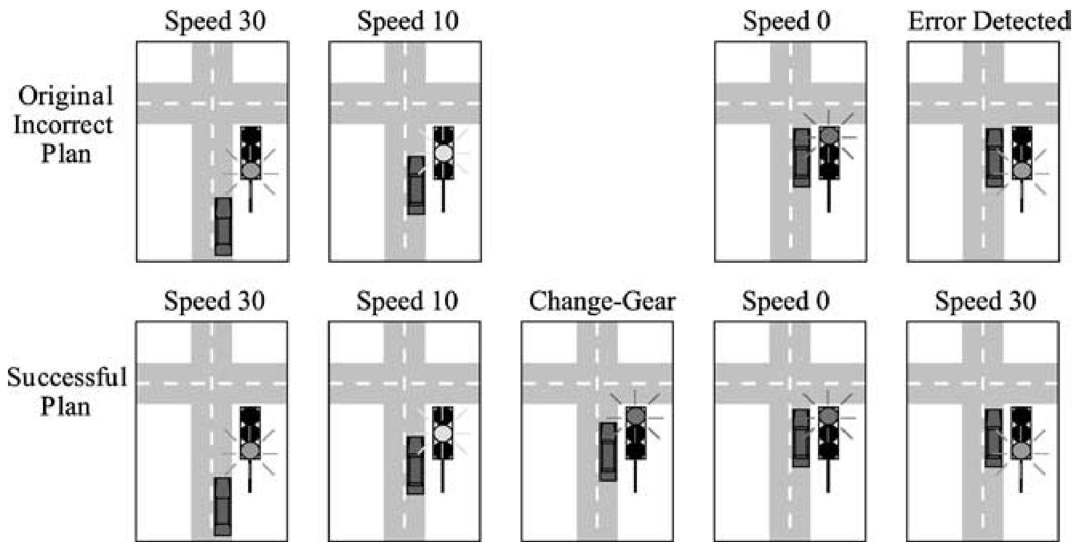


FIGURE 7. Differences between plans used for credit assignment.

shows an example where the agent is unaware that it must change gear before stopping the car or it will stall.

By comparing the successful plan to the incorrect plan, IMPROV concludes that the preconditions for `Change-Gear` should be generalized (so that an operator will be chosen in the future), while the preconditions of `Set-Speed 0` should be specialized (so that it is only chosen once the agent has changed gear). This approach can more accurately locate the incorrect operator than existing incremental approaches because IMPROV has access to more information in the form of the successful plan. Traditional approaches that only consider a single incorrect plan during learning are forced to rely on a fixed bias. For example, temporal difference methods assign most blame to the final step of a plan (`Set-Speed 0`). It is very difficult for such a system to discover that the true correction is earlier in the plan, while still maintaining the final `Set-Speed 0` operator, which is required in any successful plan.

2. *Credit assignment—How are the operators incorrect?* Having determined which operators are incorrect, IMPROV must decide how to specialize or generalize the agent's precondition knowledge. IMPROV achieves this by inductively learning the correct operator precondition knowledge. The inductive, symbolic category learner, SCA2 (Miller 1991, 1993; Pearson 1996), is trained on each instance of an operator succeeding or failing, for a particular state and goal (see Figure 6). IMPROV's behavior is not tied to this choice of inductive learner and an alternative category learner could have been used here.

   SCA2 represents its classification knowledge as production rules. Initially these rules are very general, testing only a few features from the training instances. As learning progresses more specific rules are acquired that test more features. When making a prediction, SCA2 searches for the most specific rule that matches the test instance. During training, a new rule is learned based on the most specific match found plus one additional feature. The ability to select which features are really relevant to the correctness of a plan determines the quality of the final learning (Pearson 1996).

   IMPROV analyzes the set of training examples to determine the state features that are most likely to have caused the success or failure of the operator. IMPROV biases the

induction toward features that are present in positive instances, but missing or different in negative instances. No matter how good the inductive learner is, there is insufficient information from just the initial negative instance to determine the cause of the failure. IMPROV benefits by delaying it's learning until it has found a successful plan and therefore avoids making an incorrect early induction. As the induction is based on a set of instances, we call this *k-incremental* learning. The *k* refers to the size of the set of instances passed to the learner during training and the learning is still *incremental* as the set of instances only increases until a successful plan is discovered. *K* will vary from problem to problem, as *k* is the number of trials of an operator before a success is discovered. However, as the number of instances considered during learning does not grow over the life of the agent, the learning is still incremental. This weak inductive learning can also be made stronger by the addition of domain-specific heuristics.

SCA2's inductive learning method leads to a number of compelling functional properties:

a. Learning is incremental: Instances are discarded after training, so the time to train on an instance is independent of the total number of training instances presented to the learner.
b. Performance does not slow with learning: Because SCA2 searches its rule base from specific to general rules, the time to make a prediction remains constant or decreases over time because new rules are matched sooner in the search. This is potentially offset by increasing match cost as the number of rules grows, but our experiments with IMPROV have not shown any slowdown as more rules are learned.
c. Expressive concepts can be represented: SCA2 can represent complex disjunctive and conjunctive categories by combining multiple prediction rules, each of which represents a section of the spaces of instances.
d. Additional knowledge can guide the learning: As SCA2 is fully encoded within a general problem-solving architecture, arbitrary knowledge and reasoning can be included in the critical feature selection step (see Figure 21).
e. Tolerant of noise: Incorrect prediction rules (based on noisy training instances) can be overridden by learning more specific rules that mask the earlier general rules giving SCA2 a degree of tolerance for noise. This tolerance is evaluated and described in more detail in other work (Pearson 1996).

3. *Changing the domain knowledge*: IMPROV's procedural access to the agent's domain knowledge means that the agent cannot directly examine and modify the incorrect knowledge. Instead of searching its rule base for the incorrect knowledge (a potentially expensive process), IMPROV learns additional rules that correct the decision about which operator to select (see Figure 6). Operator preconditions are specialized by learning rules that indicate the operator should not be chosen. Preconditions are generalized by learning additional rules for when the operator should be selected.

   The preconditions of an operator determine whether it is included in a particular operator hierarchy. An operator can be added to the hierarchy by generalizing its preconditions, or an operator can be removed by specializing its preconditions.

For example, the agent's initial knowledge in Figure 8(a) is incorrect as the `Shift-Up` operator is included in the implementation strategy (or plan) to achieve `Set-Speed 20`. The correct operator, `Shift-Down`, is included in the final hierarchy (Figure 8(b)), by generalizing its preconditions so that it is chosen when decelerating to 20 mph. At the same time, the incorrect `Shift-Up` operator is removed by specializing its preconditions so that it is not

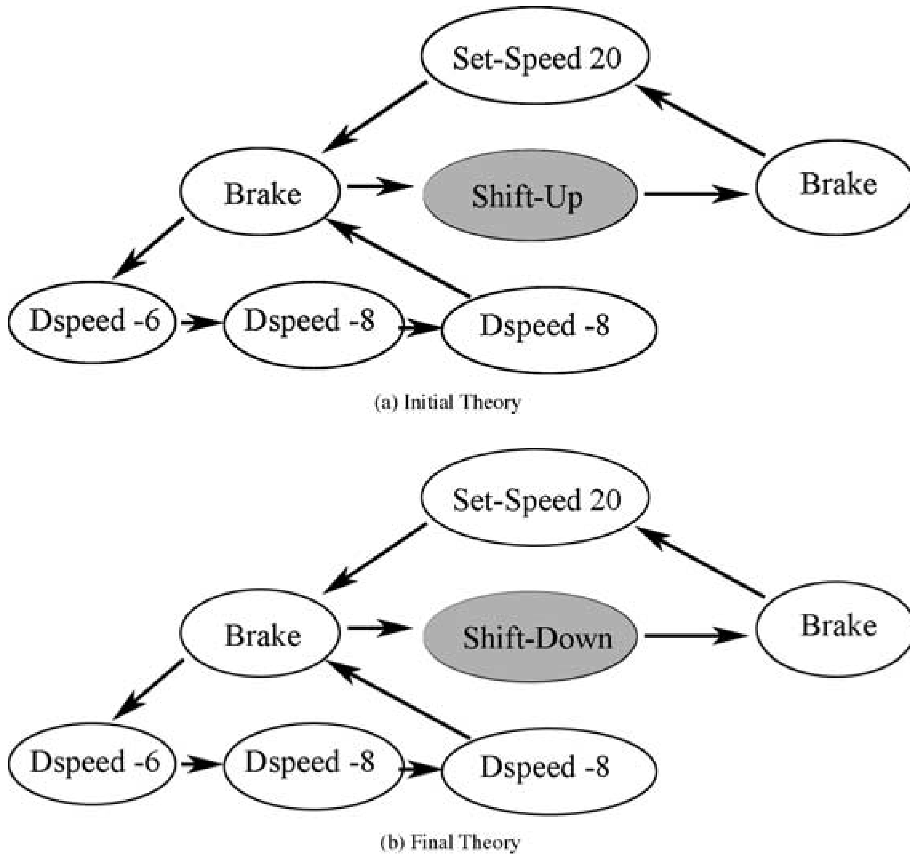


FIGURE 8. Correcting preconditions for Shift-Up and Shift-Down.

```
IF   goal(set-speed 20)
and  speed(greater-than 20)
and  gear(high)

THEN choose-operator(shift-down)
```

(a) Generalizing Shift-Down

```
IF   goal(set-speed 20)
and  speed(greater-than 20)
and  gear(high)
and  choose-operator(shift-up)

THEN reject-operator(shift-up)
```

(b) Specializing Shift-Up

FIGURE 9. New rules added to correct existing precondition knowledge.

chosen when decelerating. Examples of rules that might be learned to produce these changes are shown in Figure 9.

6. CORRECTING OPERATOR EFFECTS

IMPROV corrects the planning knowledge that models the effects of external actions. The corrected planning knowledge is then used for subsequent planning and to learn and

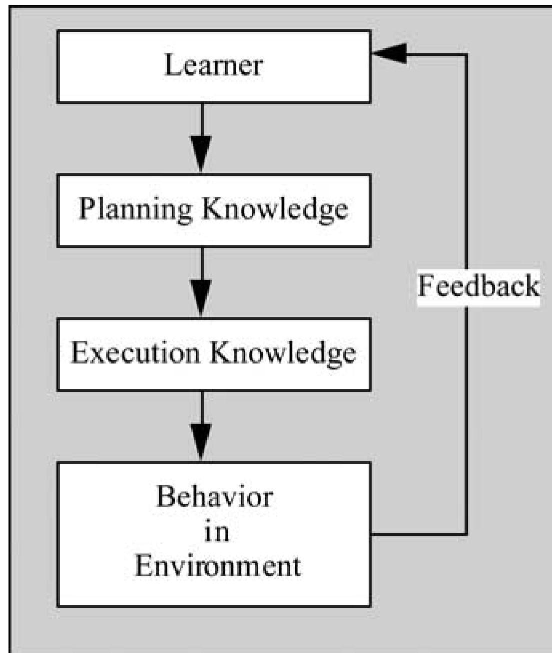


FIGURE 10. Correcting external behavior by correcting planning knowledge.

correct execution knowledge, as shown in Figure 10. Thus, the task for IMPROV is to learn the correct operator effects.

IMPROV corrects planning knowledge for the effects of operators by correcting the preconditions of a sequence of more primitive operators at the next lower level of the operator hierarchy. To see how operator preconditions can be used to correct operator effects, consider the example shown in Figure 11.

In this example, the agent's initial knowledge models the effects of pressing the brake pedal as producing a faster initial rate of deceleration than actually occurs (−4 rather than −6). To learn the correct effects, IMPROV specializes the preconditions of `DSpeed -6` and generalizes the preconditions of `DSpeed -4`. This is completely analogous to the earlier example (Figure 8), with the correction being applied to a different level in the operator hierarchy. It is important to realize that the lowest level of operators only represent planning knowledge; the agent's external actions are the same in both cases, which is to press the brake pedal when the `Brake` operator is chosen. By changing the sequence of single-effect operators that are chosen in implementing `Brake` during planning, the agent's model for the effects of braking is changed, so that the agent expects braking to occur more slowly. Then, when the agent replans, it will select the `Brake` operator earlier, as it expects slower braking. This approach allows IMPROV to represent and learn complex models of sequential or conditional effects that occur over time. For example, IMPROV can learn the multiple, dynamic effects of braking shown in Figure 12.

This figure shows the rate of deceleration (`DSpeed`) and pressure on the brake pedal (`Brake-Pressure`) over time, in response to a *single* external action (to press the brake pedal). IMPROV models this sequence of effects as a series of single-effect operators and learns the correct preconditions for those operators. Figure 13 shows the operator hierarchy after learning (`T` operators indicate when time advances).

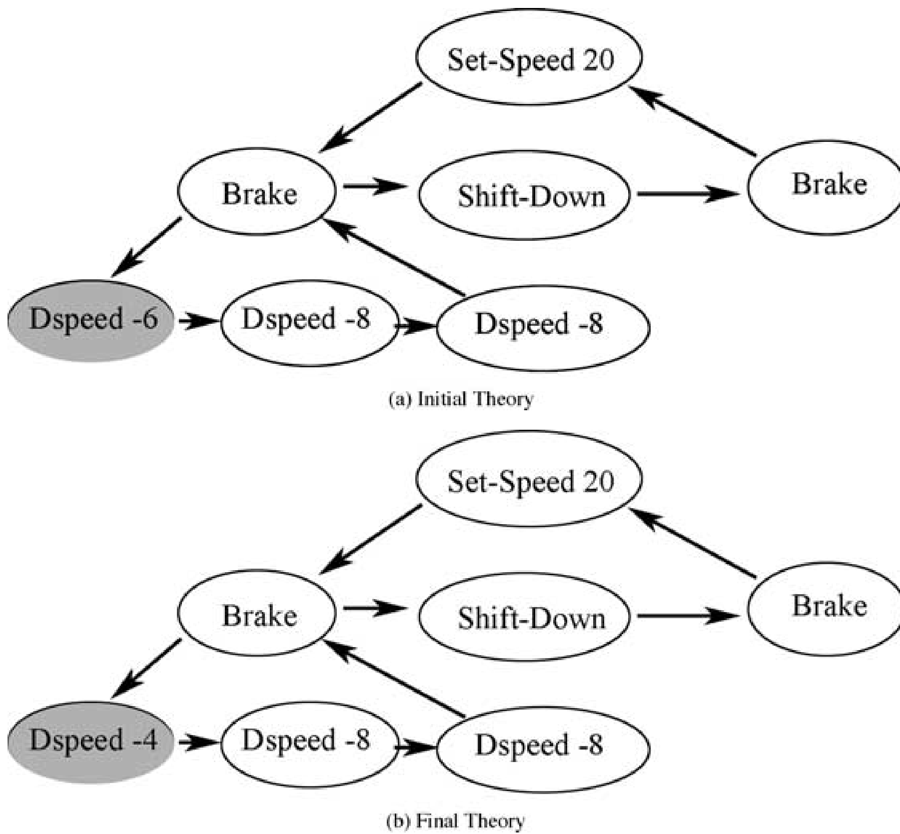


FIGURE 11. Correcting preconditions for DSpeed −4 and DSpeed −6.

Existing systems that learn the effects of operator actions are unable to represent or learn this type of sequential effect. IMPROV's approach to learning operator effects, by correcting precondition knowledge at a lower level, is guaranteed to terminate at the single-effect level. This is because single-effect operators only manipulate a single symbol (e.g., `DSpeed`) and

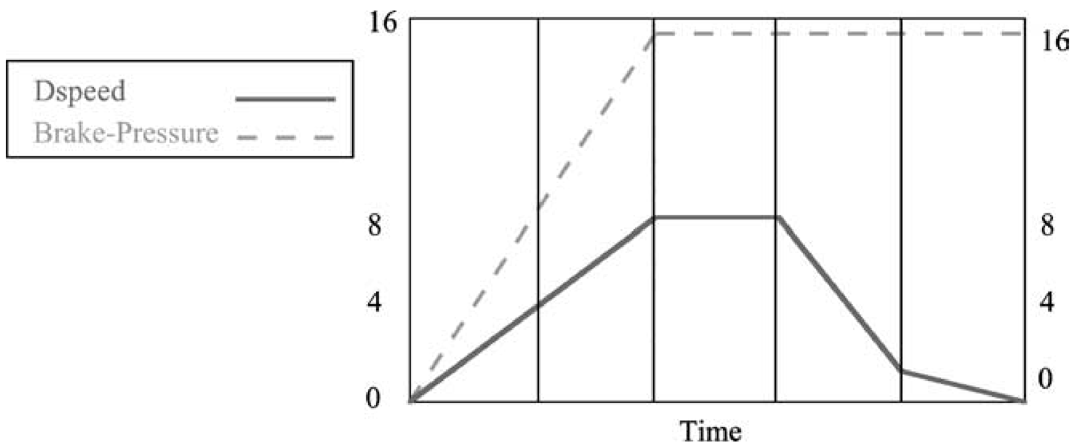


FIGURE 12. Correct model for the effects of braking.

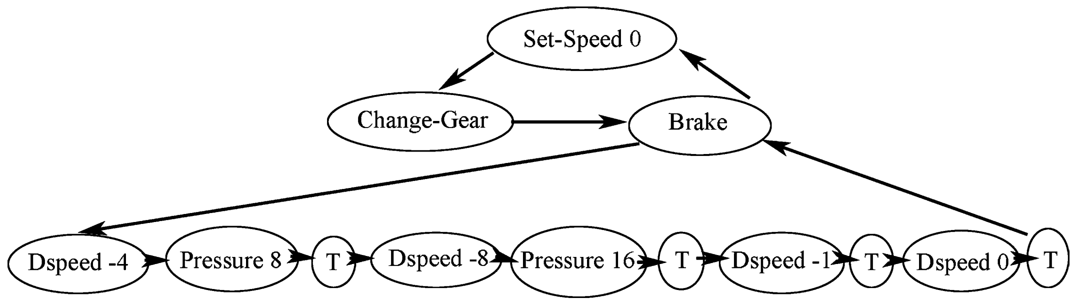


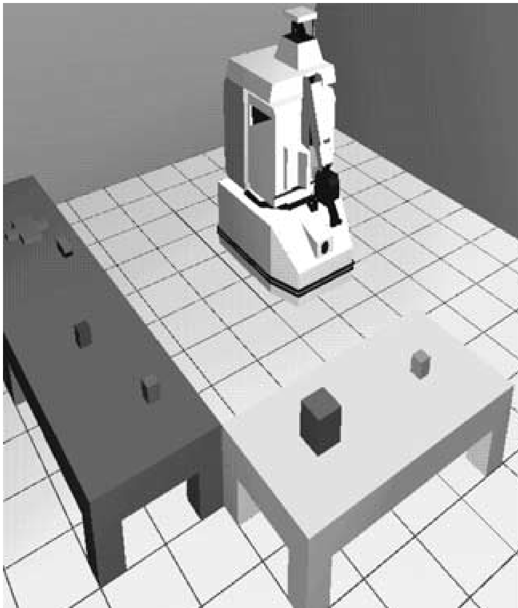
FIGURE 13. IMPROV's final representation for the effects of braking.

therefore the knowledge about the effects of these operators is guaranteed to be correct. The question is whether or not to include one of these operators in the effects of a motor-level operator, instead of changing the effects modeled by the single-effect operators; a decision that is based on the precondition knowledge of the `DSpeed` operators.

7. EVALUATION

We have evaluated IMPROV on two test domains: a simulated robotic manipulation task and a simulated car driving domain (see Figure 14).

The task in the robot domain is to align blocks on tables. The blocks have different characteristics and the agent must learn which of the blocks can be successfully moved. The robot can move around the room and has a single gripper. The task in the driving domain is to



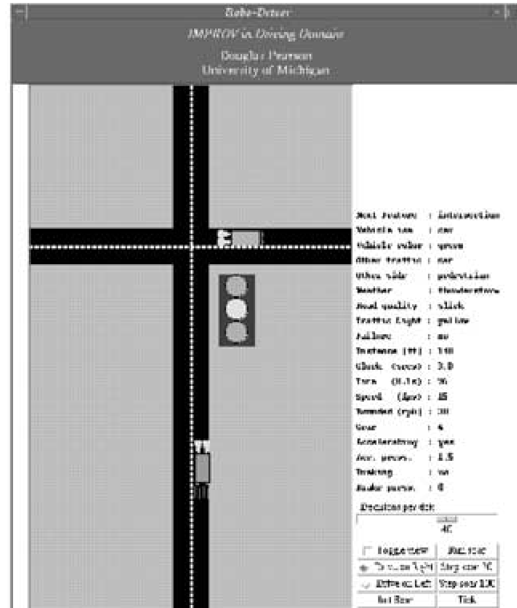


FIGURE 14. Robotic and driving domains.

Attributes	Number of possible values	Sample Initial Knowledge (+3x2)	Sample Final Knowledge		
Distance	5	Close	Close	Close	Close
Speed	10				
My-vehicle-isa	10		Car		
My-vehicle-color	10				
My-road-sign	2	Signal	Signal	Signal	Signal
My-light-color	3		Red	Green	Green
Gear	4				
Weather	10				
Road-Quality	10				
First-road-user-isa	10			Police	Ambulance
Sidewalk-user-isa	10				
Other-road-sign	2				
Operator:		set-speed(30)	set-speed(0)	set-speed(0)	set-speed(0)

FIGURE 15. Example domain knowledge in driving domain experiment.

successfully cross an intersection. There are other cars and pedestrians in the environment and processes (such as traffic lights) that change independently of the agent's actions. The agent must learn the correct procedure for crossing the intersection. For example, the agent should stop for red lights or police cars. The agent starts with sufficient initial knowledge to build a plan that it believes will succeed. This knowledge ensures that the agent is not performing a blind search, either for a correct plan or for the correct operator knowledge. In both cases, the search is usefully biased by the agent's initial knowledge. The experiments were not designed to compare IMPROV's learning rate or accuracy to that of other systems, but rather to evaluate the scope of IMPROV's learning and demonstrate its ability to learn in environments with a wide range of challenging properties. The key point to take from each experiment is the demonstration that IMPROV could perform in a particular type of environment, while using a deliberate, knowledge-intensive, and yet efficient learning method. Figure 15 shows an example of a learning experiment.

The attributes considered during learning are shown on the left, along with the range of values each attribute can take. Next, a part of the initial knowledge given to the agent is shown (i.e., that the `set-speed 30` operator should be chosen when the distance is close and the road sign is a traffic signal). Finally, an example of a target theory is shown in the third column. In this experiment, the agent must learn three exception cases to the initial theory's general rules. The target operator preconditions consist of three disjunctive terms, each containing two additional conjunctive terms that are missing in the initial knowledge (the term distance(close) is already present). This example is labeled as $+3 \times 2$. The $+$ indicates that the initial theory is overgeneral and must add three disjunctive terms (each of two conjuncts) to reach the target theory. Overspecific initial theories are the converse, for example -3×2 would mean the agent started with the target theory shown and had to learn the initial theory. The robot domain contains a similar number of attributes and has test cases formulated in the same manner. Each experiment reflects the average results from 10 runs and each test case is designed so that the agent's initial, incorrect knowledge will lead to a failure if the agent does not learn. This makes it easier to identify the effect that learning has

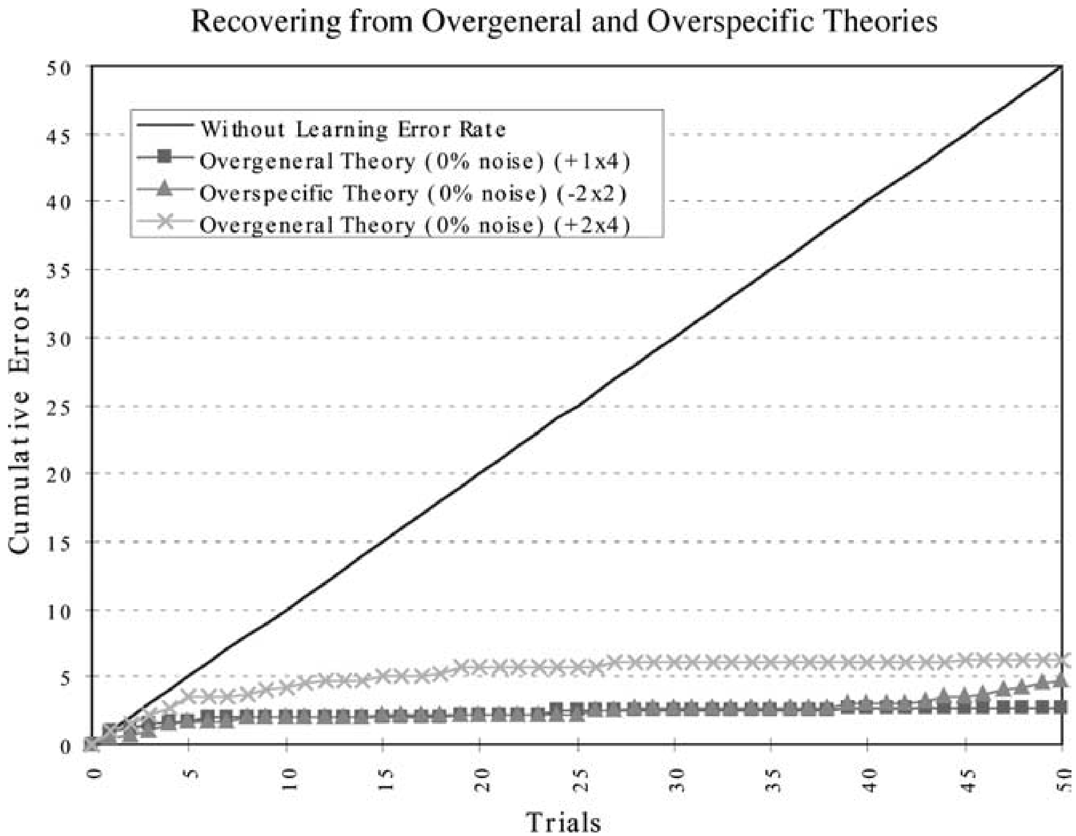


FIGURE 16. Recovering from overgeneral and overspecific theories.

on the agent's ability to perform the task. Without any learning, every trial would lead to an error.

7.1. Coverage of Classes of Errors

IMPROV has been demonstrated correcting domain knowledge that initially included overgeneral and overspecific operator preconditions, incomplete operator effects, and extraneous operator effects. IMPROV has not been tested on, and has no ability to learn, when there are completely missing operators.

Figure 16 shows an example of this behavior. The graph shows the cumulative number of errors made by IMPROV over the course of 50 trials, for a range of target domain theories. The diagonal line is a reference showing the number of errors that would occur without learning. This graph simply demonstrates that IMPROV can correct overgeneral and overspecific initial theories and quickly converges to a reasonable approximation of the correct theory, resulting in few total errors.

7.2. Large State Spaces

The state space for the driving domain exceeds a billion distinct states, with certain parts of the state (the other agents) being added and removed to produce dynamic changes in the

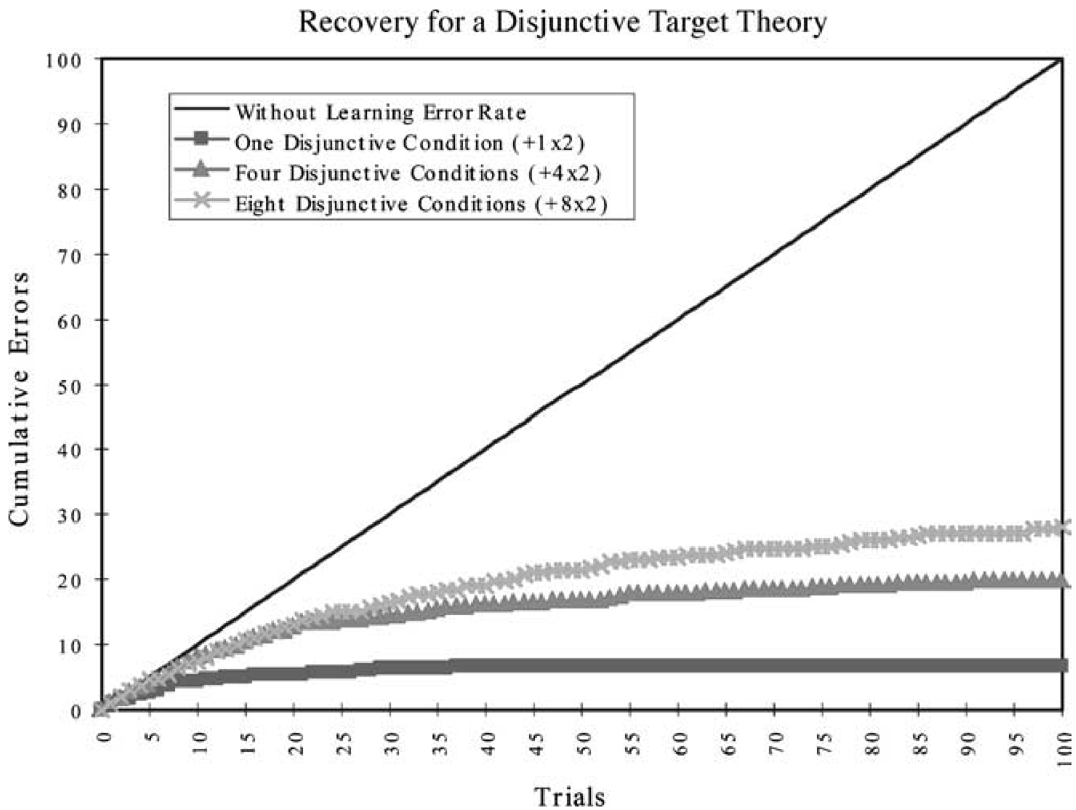


FIGURE 17. Learning disjunctive precondition knowledge.

representation. Neither of the goal spaces is particularly large, but the robot domain allows for in excess of 100 different goal states.

Each domain includes many conditional actions and a number of sequential effects that occur over time as a result of a single external action. We tested IMPROV on tasks with disjunctive preconditions that occur when actions can apply in a range of disjoint states. Figure 17 shows the cumulative errors made as IMPROV learns to correct domain knowledge that includes an increasing number of missing disjunctive terms. Disjunctive preconditions can present difficulties for some existing learning methods and this graph shows that while learning is more difficult as the number of disjuncts increases, IMPROV quickly converges toward the correct knowledge.

7.3. Environmental Complexities

The driving domain included multiple agents and other external processes that change the environment asynchronously and without action by the agent. This domain also contains actions that cannot be reversed, requiring the agent to learn across multiple training episodes. The agent was only provided limited sensing of the environment, with noise being added to its inputs and delays in the feedback that an action would lead to success or failure. Additionally, the underlying physics of the domain could be modified after training IMPROV for a period in the environment. An example of this is when braking takes more time as tires wear out or when part of a robot starts to malfunction. Figure 18 shows an example where the environment changes after 50 trials. The unexpected change causes the agent to make more errors, but it

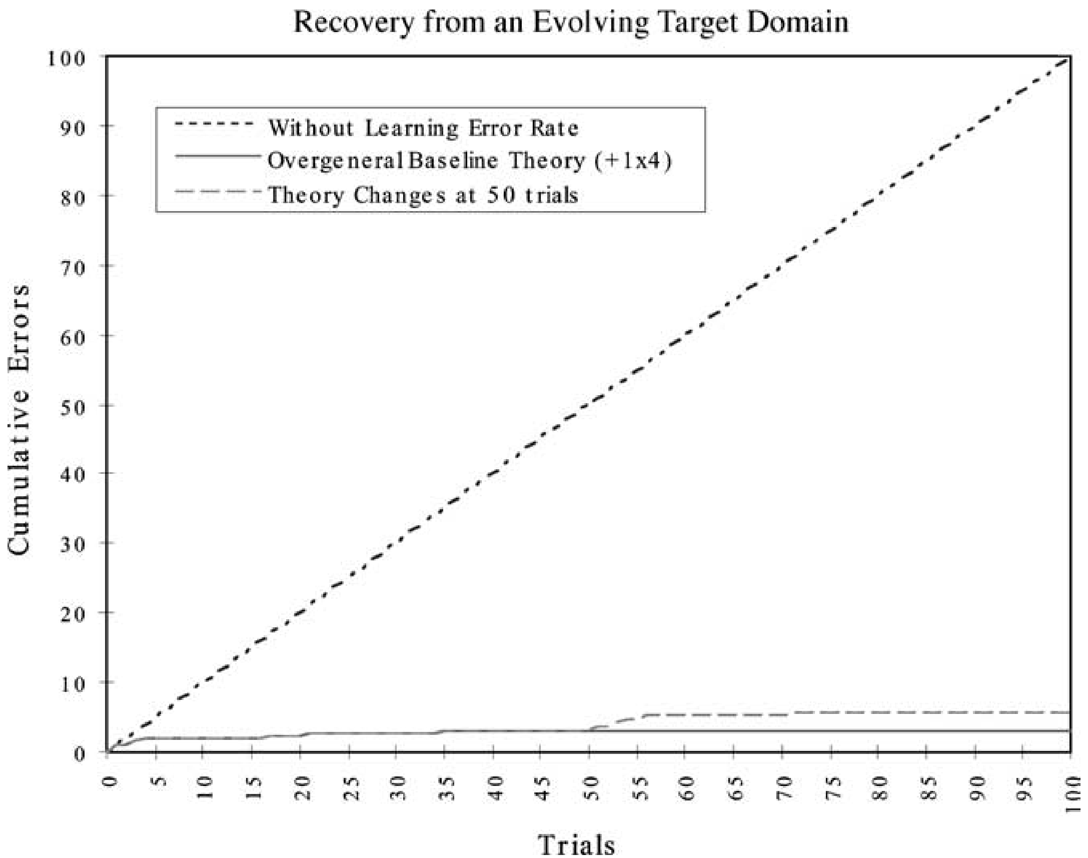


FIGURE 18. Learning as the target knowledge changes over time.

then adjusts to the new theory. The baseline theory shows the behavior when the target theory remains constant.

7.4. Constraints on the Agent

The driving domain requires the agent to act within a fixed time limit as the environment is changing asynchronously. The agent always had a fixed amount of time for processing. This time was sufficient to solve the problems as long as the agent did not become slower as a result of learning. Figure 19 shows the CPU time per trial while IMPROV is performing on the previous evolving domain problem (Figure 18). The first spike includes the time for the agent to build an initial plan. Later spikes occur when a correction had to be made to the agent's domain knowledge. The time spent on each correction remains constant or decreases as the agent learns and the theory becomes more complex. This is in contrast to many symbolic machine learning algorithms that become slower as the theory they are learning grows more complex. This result helps to support the hypothesis that limiting an agent to procedural access to its knowledge leads to an efficient correction method.

7.5. K-Incremental Learning

IMPROV's ability to accurately assign credit for successes and failures during learning is improved by delaying learning until a successful plan has been found. Figure 20 demonstrates

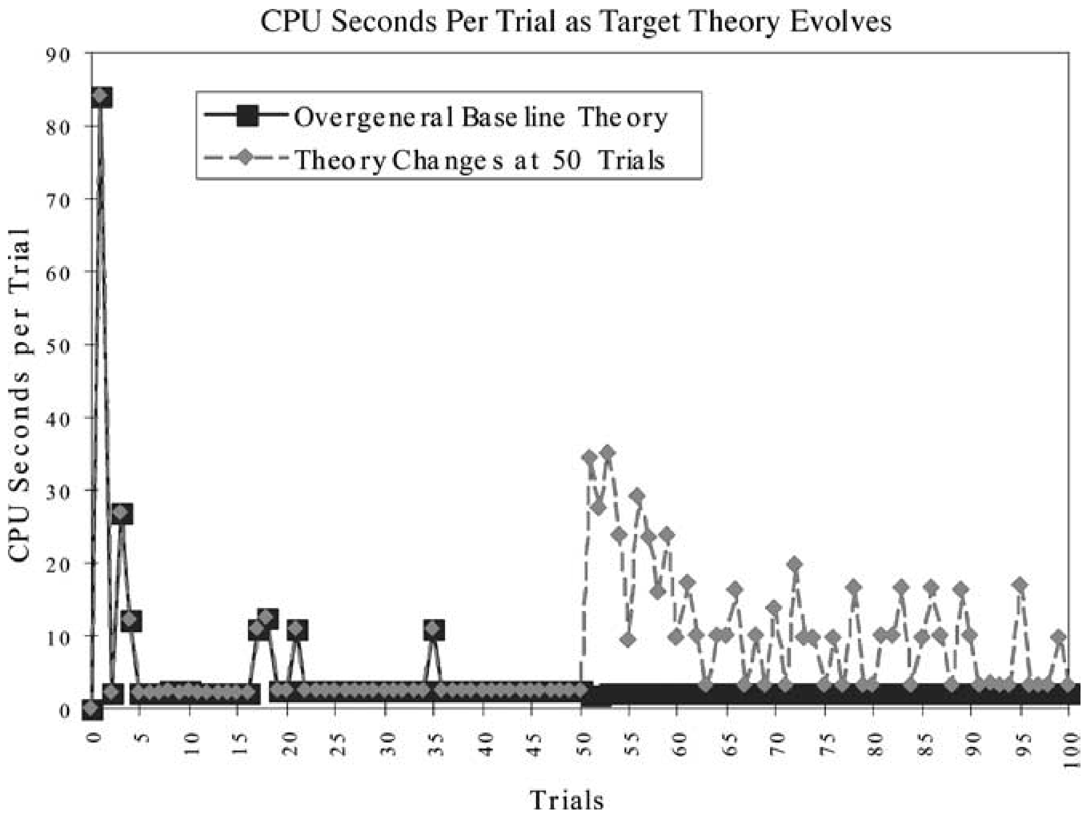
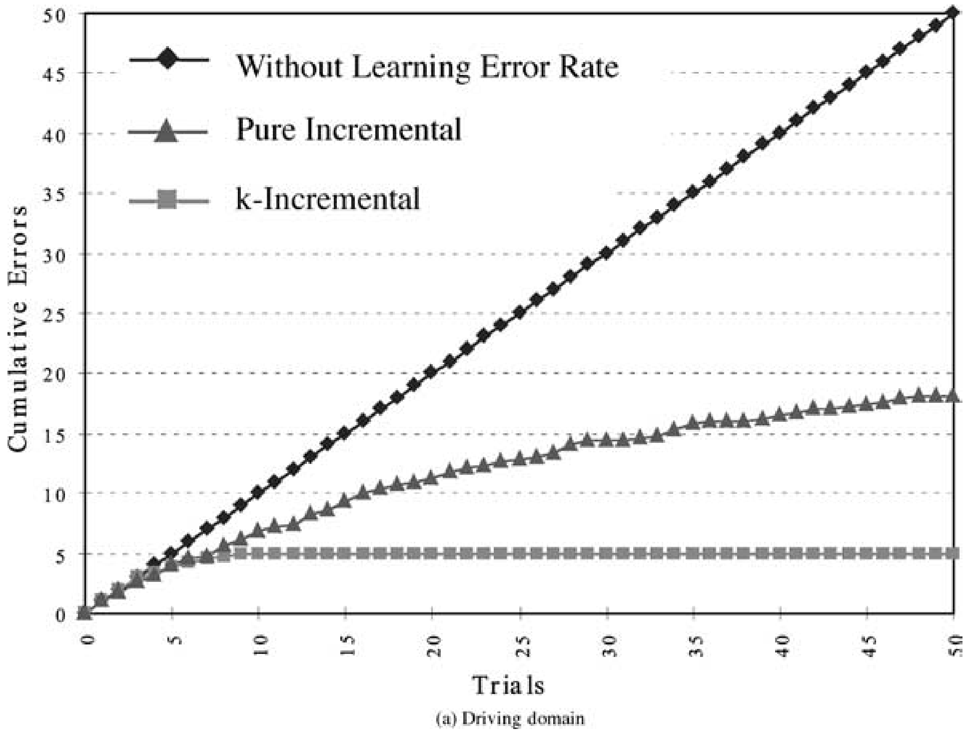


FIGURE 19. CPU time per trial as the agent learns.

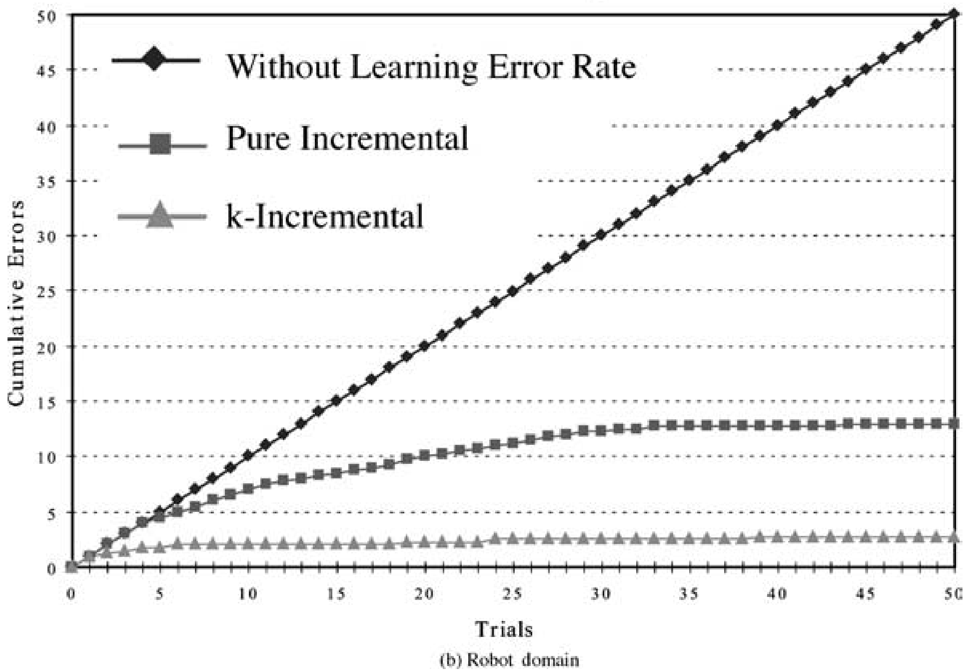
the benefit of using this deliberate, analytic k-incremental approach over a pure incremental learner. IMPROV was modified to train immediately after seeing each individual instance, rather than waiting and training on a set of instances, thereby simulating a pure incremental system. The difference in the resulting error rates is substantial, confirming that the accuracy of the learning benefits from delaying learning until credit can be assigned more accurately.

7.6. Knowledge-Directed Learning

IMPROV represents a weak, general-purpose method for learning planning knowledge in a range of challenging environments. IMPROV's symbolic knowledge representation in a general-purpose architecture simplifies the addition of extra knowledge to guide learning. The additional knowledge could either come from adding new procedurally accessed rules or from an unstructured source, such as an instructor. In Pearson and Huffman (1995), we showed how instructions presented in English could be used to guide learning (e.g., "Think about your speed" could be used to guide the inductive learning phase). Knowledge can be added to guide IMPROV's error detection, replanning, and learning phases. Error detection was improved by adding an early indication that an action was about to lead to failure. For example, in the driving domain this could be an instructor who shouts "Look Out!" prior to the failure. Replanning was improved by adding knowledge that selecting speed 0 was

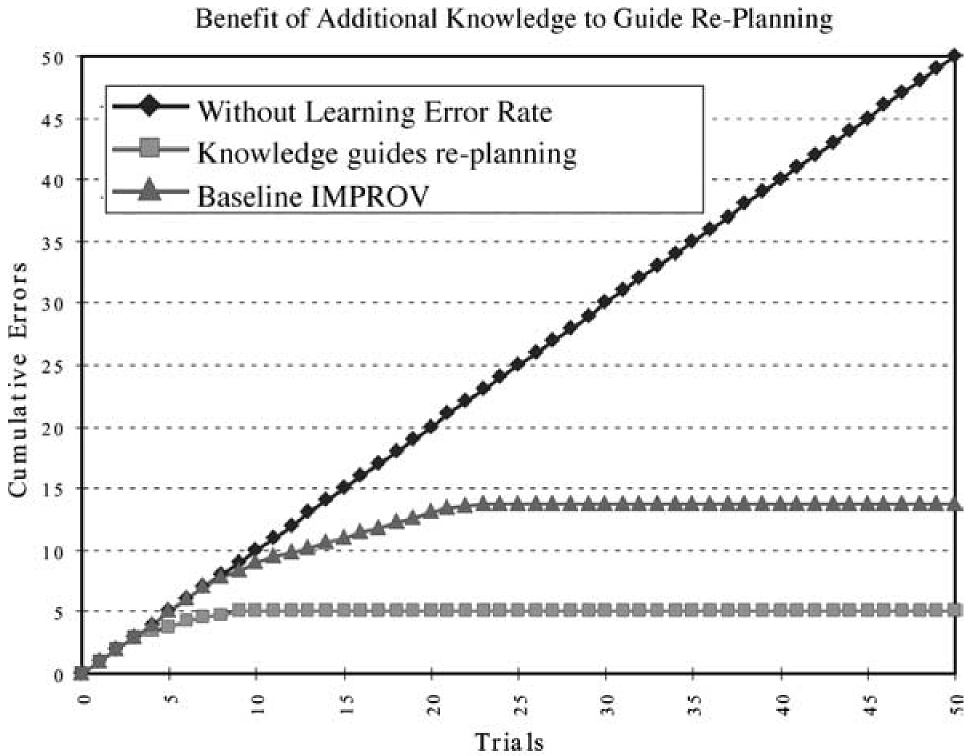


(a) Driving domain

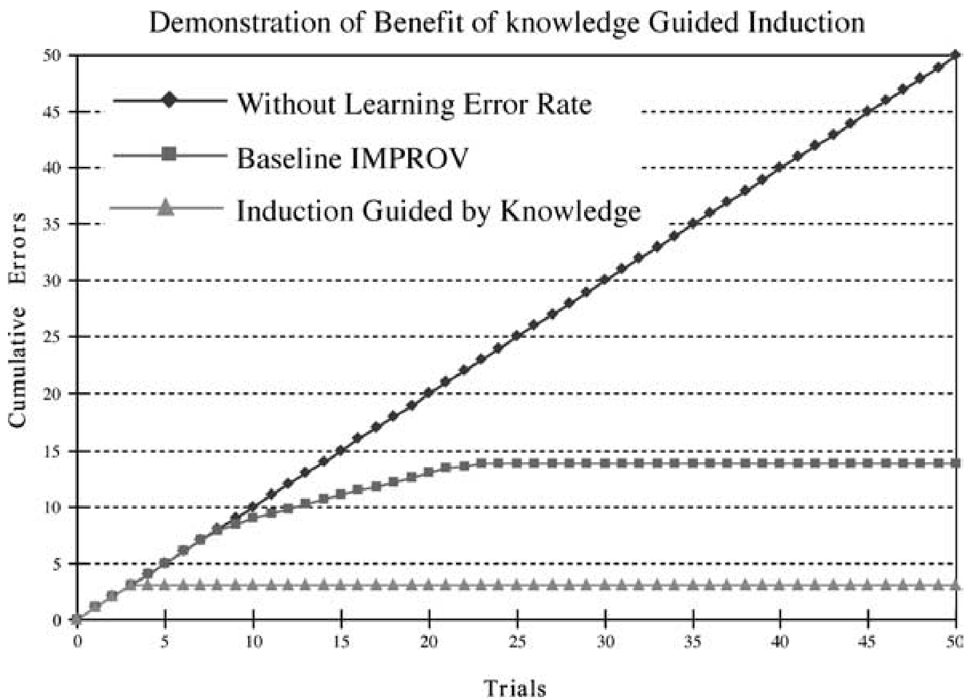


(b) Robot domain

FIGURE 20. Comparison of k-incremental learning with pure incremental learning.



(a) Guiding replanning



(b) Guiding induction

FIGURE 21. Benefits of adding knowledge to IMPROV.

functionally different from selecting other speeds (Figure 21(a)). Inductive learning was improved by biasing the learner toward aspects of the domain that are particularly relevant (e.g., the state of the traffic light when driving) (Figure 21(b)). This is a much simpler task than correctly specifying the agent's behavior in the environment, which is to say that knowing that traffic lights are important is much easier than knowing how to cross a busy intersection.

The important point in these experiments is not the degree of improvement in the agent's performance, which is naturally a function of the quality of the additional knowledge. Instead, the important point is that additional knowledge can be easily added to produce this improvement in performance, turning a general learning method into a strong learner and showing that limited procedural access to a compact, intentional representation can still support strong deliberate learning while allowing the agent to function in complex environments.

REFERENCES

BAFFES, P., and R. MOONEY. 1993. Symbolic revision of theories with m-of-n rules. *In* Proceedings of the International Join Conference on Artificial Intelligence, pp. 1135–1140, Chambéry, France.

BOOKER, L. B., D. E. GOLDBERG, and J. H. HOLLAND. 1989. Classifier systems and genetic algorithms. Artificial Intelligence, **40**:234–282.

FIKES, R. E., and N. NILSSON. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, **2**:189–208.

GIL, Y. 1992. Acquiring domain knowledge for planning by experimentation. Ph.D. Thesis, Carnegie Mellon University.

GIL, Y. 1993. Efficient domain-independent experimentation. *In* Proceedings of the International Conference on Machine Learning, pp. 128–134, Amherst, MA.

GIL, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. *In* Proceedings of the International Conference on Machine Learning, pp. 87–95, New Brunswick, N.J.

HOLLAND, J. H. 1986. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *In* Machine Learning: An Artificial Intelligence Approach, volume II. Morgan Kaufmann, Los Altos, CA.

LAIRD, J. E., A. NEWELL, and P. S. ROSENBLOOM. 1987. Soar: An architecture for general intelligence. Artificial Intelligence, **33**(1):1–64.

MILLER, C. M. 1991. A constraint-motivated model of concept formation. *In* The Thirteenth Annual Conference of the Cognitive Science Society, pp. 827–831, Hillsdale, NJ.

MILLER, C. M. 1993. A model of concept acquisition in the context of a unified theory of cognition. Ph.D. Thesis, The University of Michigan.

OURSTON, D., and R. J. MOONEY. 1990. Changing the rules: A comprehensive approach to theory refinement. *In* Proceedings of the National Conference on Artificial Intelligence, pp. 815–820, Boston, MA.

PAZZANI, M. J. 1988. Integrated learning with incorrect and incomplete theories. *In* Proceedings of the International Machine Learning Conference, pp. 291–297, Ann Arbor, MI, USA.

PAZZANI, M. J., C. A. BRUNCK, and G. SILVERSTEIN. 1991. A knowledge-intensive approach to learning relational concepts. *In* Proceedings of the Eighth International Workshop on Machine Learning, pp. 432–436, Ithaca, NY.

PEARSON, D. J. 1996. Learning procedural planning knowledge in complex environments. Ph.D. Thesis. University of Michigan.

PEARSON, D. J., and S. B. HUFFMAN. 1995. Combining learning from instruction with recovery from incorrect knowledge. *In* Machine Learning Conference Workshop on Agents that learn from other agents. Available from www.sunnyhome.org/pubs/mlw95.html.

PEARSON, D. J., and J. E. LAIRD. 1999. Toward incremental knowledge correction for agents in complex environments. *In* Machine Intelligence, volume 15. Oxford University Press, New York.

QUINLAN, J. R. 1990. Learning logical definitions from relations. Machine Learning, **5**(3):239–266.

RUMELHART, D. E., G. E. HINTON, and R. J. WILLIAMS. 1986. Learning internal representations by error propogation. *In* Parallel Distributed Processing, volume 1. MIT Press, Cambridge, MA.

SAMUEL, A. L. 1959. Some studies in machine learning using the game of checkers. IBM Journal on Research and Development, **3**:210–229.

SHEN, W., and H. A. SIMON. 1989. Rule creation and rule learning through environmental exploration. *In* Proceedings of the International Joint Conference on Artificial Intelligence, pp. 675–680, Detroit, MI.

SUTTON, R. S. 1988. Learning to predict by the methods of temporal differences. Machine Learning, **3**:9–44.

TESAURO, G. 1992. Temporal difference learning of backgammon strategy. *In* Proceedings of the Ninth International Conference on Machine Intelligence, pp. 451–457, Aberdeen, Scotland.

WANG, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. *In* Proceedings of the Twelfth International Conference on Machine Learning, pp. 549–557, Tahoe City, CA.

WANG, X. 1996. Learning planning operators by observation and practice. Ph.D. Thesis, Carnegie Mellon University.

WATKINS, C. J. C. H., and P. DAYAN. 1992. Technical note: Q-learning. Machine Learning, **8**:279–292.