Machine Learning, 7, 7–44 (1991)

An Incremental Deductive Strategy for Controlling Constructive Induction in Learning from Examples

RENÉE ELIO (REE@CS.UALBERTA.CA)
Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada T6G 2HI

LARRY WATANABE (WATANABE@CS.UIUC.EDU)
Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada T6G 2HI

Editor: Pat Langley

Abstract. This article describes LAIR, a constructive induction system that acquires conjunctive concepts by applying a domain theory to introduce new features into the evolving concept description. Each acquired concept is added to the domain theory, making LAIR a closed-loop learning system that weakens the inductive bias with each iteration of the learning loop. LAIR's novel feature is the use of an incremental deductive strategy for constructive induction, reducing the amount of inference required for learning. A series of experiments manipulated features of learning tasks to assess this incremental method of constructive induction relative to an uncontrolled constructive induction process that extends each example description with all derivable features. These learning tasks differed in global characteristics of the domain theory, the training sequence, and the percentage of irrelevant features in the example descriptions. The results show that LAIR's constructive induction approach saves considerable inferencing effort, with little or no cost in the number of examples needed to reach a learning criterion. The experimental results also underscored the importance of viewing a domain theory as a search space, identifying several factors that impact the deductive and inductive aspects of constructive induction, such as concept definition overlap, density of features, and fan-in and fan-out of inference chains. The paper also discusses LAIR's operation as a pac-learner and its relation to other constructive induction techniques.

Keywords. Constructive induction, learning from examples, knowledge acquisition, domain knowledge and learning

1. Introduction

Learning from examples is an important form of induction that has been widely studied in both humans and machines. In recent years, machine learning researchers have explored the role of domain knowledge in different types of inductive tasks. Knowledge of a domain may not only influence how easily a learner acquires new concepts in that domain, it in fact may be necessary to accomplish the learning task. In this article, we describe a learning system that combines deductive and inductive techniques to apply and extend a domain theory for learning concept definitions from examples.

In learning from examples, the goal is to acquire a concept description that can be used to predict whether or not a novel item is an instance of the concept. This has been formalized in theoretical learning as probably approximate correct learning, or *pac-learning* (Valiant, 1984). A pac-learner is said to learn from examples if it can, in a feasible (polynomial) amount of time, find with high likelihood a reasonably accurate rule (Kearns, Li, Pitt, & Valiant, 1987). Reasonably accurate means that the rule found by the algorithm

will be able to predict future events drawn from the same distribution on which it has learned, with controllable error.

Within a framework for incremental learning from examples, each new training example typically prompts some revision of the current concept description. If the instance is a positive example of the concept, but the current concept description incorrectly rejects it, the concept description is too specific and must be generalized so that it matches the new instance. If the current example is a negative instance but matches the current concept description, then the concept description is too general and is specialized.

As long as transformation operators manipulate only the features present in the example descriptions, the instance language effectively defines the space in which the concept description lies. Generalization operators may drop features from the concept description, replace constants with variables, and so forth. Specialization operators can make the description more specific by adding a feature or constraining variables to accept only certain values. The learning task involves identifying which combination of features or constraints on features within this space best covers the positive training items and but not the negative items. Rendell (1986) has called this the task of *selective induction*, and Michalski (1983) refers to these kinds of inductive operators as *selective transformation rules*.

When a learner introduces new features from outside the instance language, the space of possible concept descriptions is altered. This kind of transformation, formalized by Michalski (1983) as the constructive generalization (or *constructive induction*) rule, uses background knowledge or a domain theory to derive additional information about an example. For example, suppose that a learner's current concept description for *chair* is *has-part (object,legs) & has-part (object,seat) & sit-on (people,object)*. If the learner also knows *has-part (object,seat) & sit-on (object,people)* $\rightarrow$ *function (object,support)*, then Michalski's constructive induction rule could transform this description to *has-part (object,legs) & function (object,support)*.

This type of transformation of the feature space is important for several reasons. First, many learning problems are very difficult without a change in representation. Clearly, if the concept description lies outside the description space that is defined by the instance language, then it can only be learned by somehow transforming or extending that space. Second, the availability of a domain theory and a constructive induction operator eases the teaching task. Rather than requiring the teacher to present all relevant and potentially relevant features in the examples, the system can use its domain theory to derive the relevant features for the task at hand, and incorporate these features into the concept description even if they were not explicitly included in the examples.

Michalski (1983) identified several specific versions of the constructive induction rule, but its controlled use as an inductive operator has not been widely studied. For example, his INDUCE system augments each example description with all the derivable features, so essentially there is no control. Clearly, this approach will derive many additional features that could be used to generalize or specialize the concept description, but how many of these will be relevant to the learning task? Simply put, the invocation and guidance of constructive induction remain unresolved issues.

In this paper we investigate these issues in the context of an incremental learning system that maintains a single conjunctive concept description in the derived description space and updates it with each new example presented.[1] The concept description must be general

enough to cover new positive instances and specific enough to reject new negative instances. Thus, the learning requirements of this type of incremental learning are very well-defined, and inductive operators can be formulated to meet these requirements.

We have taken advantage of these requirements to define the constructive induction process as a deductive step followed by an inductive step. Deductive rules used with the domain theory introduce new features into the learning task by extending what is known about an example description. This occurs when there is no explicit overlap of features between an example and a concept description, so that similarities and differences must be derived. The preconditions of the inductive operators then determine whether a particular feature is a valid transformation of the concept description. These operators determine a revision step's validity without access to previously-seen examples, which are forgotten as soon as they are processed. Instead, only a single positive example is retained in memory and this example is used by the inductive operators to validate concept revisions.

This integration of deductive and inductive methods implements constructive induction in a controlled fashion to meet the needs of incremental learning. We have explored this approach in a system called LAIR (Watanabe & Elio, 1987). In addition to using a domain theory to guide induction, LAIR extends its domain theory by adding to it the concept descriptions it learns. Once part of the domain theory, a newly-acquired concept description then becomes prior knowledge to support deduction and constructive induction in the next learning task. This type of learning is known as *closed-loop learning* (Michalski & Ko, 1988; Michalski & Watanabe, 1988).

In the sections that follow, we first present LAIR's representation language and the deductive and inductive rules that use the domain theory. Next, we outline the general control strategy for combining the deductive and inductive methods during learning. We then present examples that illustrate how these methods are used to acquire and apply a domain theory. After this, we describe our methodology for empirically evaluating LAIR and present data on its inductive and deductive behavior under different learning scenarios. Finally, we discuss LAIR as a pac-learner and the more general issues of this approach in relation to other constructive induction systems.

2. Representation language and domain theory

In this section, we describe how LAIR represents instances and concepts. To illustrate the important features of the representation, we use examples from Winston, Binford, Katz, and Lowry's (1983) domain theory for the concept *cup*. We then define what we mean by a domain theory.

2.1. Representing instances and concepts

Example descriptions are a conjunction of positive literals, i.e., unnegated features, with one or more arguments, such as *(covered object-1) & (light object-1) & (body object 1 body-1) & (small body-1) & (cylindrical body-1) & (contents objects contents-1) & (hot contents-1)*. In English, this description corresponds to "an object that is covered and light, that has

a body that is small, cylindrical, and that has contents which are hot." As this example illustrates, we represent relations among arguments by introducing a feature to declare the relation. Thus, *(contents object-1 contents-1)* introduces *contents-1* into the description by indicating its relation to the existing argument, *object-1*. The same constant names for corresponding objects are used in all examples. This gives LAIR the necessary matching information, so each of the literals can be viewed as features. At present, our representation language does not include any arithmetic operators or functions.

A concept description, a term we use interchangeably with "concept" and "concept definition," is a conjunction of negated and unnegated literals. We refer to the unnegated literals as *required* features and to the negated literals as *forbidden* features. Each time LAIR acquires a concept definition, it creates an implication of the form *concept description* → *concept-name*. Constants in the concept description are changed to variables in the rule. Implications built from concept descriptions incorporate a single, unnegated consequent that corresponds to the feature that is the name of the concept. Thus, the following implication, which we informally call an *inference rule*, might be constructed from a concept definition for "graspable": *(light =object) & (body =object =body-object) & (small =body-object) & cylindrical =body-object) & not (hot =body-object)* → *(graspable =object)*, where the equal sign indicates pattern-matching variables.

LAIR's representation language does not include a disjunctive operator, hence it has no explicit way to represent disjunctive concept definitions. However, it can functionally represent disjunctive concepts by having more than one definition for a concept. For example, the above definition of graspable can co-exist in the domain theory knowledge base with another definition, such as *(has-part-handle =object)* → *(graspable* =object).

2.2. Acquiring a domain theory

The domain theory consists of concept definitions in the form of inference rules for recognizing instances of the concepts, such as the examples above. At the "base" of the domain theory are *primitive features*. These are features that are not consequences of any inference rule, i.e., they have no definition. It is not necessary for LAIR to be given these primitive features, or for that matter, any domain theory to begin learning. A teacher, with a target concept in mind, can start presenting some training sequence of examples. LAIR simply adds any feature appearing in the example descriptions into the domain theory, if it does not already exist there.

LAIR acquires and extends a domain theory in the following way. When the system has learned a new concept at the end of a training sequence, it formulates an inference rule for recognizing it (using whatever name the teacher has given the concept) and adds this rule to the domain theory. We call this a *higher-level* concept to distinguish it from the primitive features. Higher-level concepts can appear in example descriptions for learning other concepts and their rule definitions are available for proof procedures that support subsequent learning and classification tasks. The important point is that LAIR does not need to have a domain theory in place in order to learn; the same mechanisms that use a domain theory to guide induction also allow the system to extend that domain theory, providing concepts that are part of other definitions are taught first.

3. Constructive induction method

This section describes how LAIR interleaves deductive processes with the inductive goals of incremental concept learning. We first provide a general overview of the constructive induction approach and then present specific details on the deductive processes and inductive operators.

3.1. Updating the concept description with limited example memory

As noted earlier, LAIR does not maintain a complete memory for previous examples. We think it is important to explore learning systems with limited memory for training items, because the number of training items required to reach some learning criterion can become quite large. Storing and reprocessing them as part of some backtracking scheme can be computationally expensive. However, a problem with limited memory for previous examples, coupled with maintaining only one concept hypothesis, is that the current concept hypothesis could become invalid with respect to forgotten examples.

Our approach to this problem is to use two types of information in the preconditions of our inductive operators. First, although LAIR does not remember all previous training examples, it must remember at least one previous positive example. This remembered positive example, which we refer to as the *remembered past positive example*, is held constant throughout the learning task. It serves as a kind of touchstone for validating changes to the concept hypothesis. For example, the inductive operator that adds a feature to the current concept description as a new required feature will do so only if that feature can be proven true of both the current positive example as well as the remembered past positive example.

The inductive operators' preconditions also place constraints on which features can be added to the concept description. Specifically, LAIR remembers whether a feature has appeared in a positive or negative example and whether it has been in previous versions of the concept description. For example, the inductive operator that adds a required feature also checks that this feature had not been previously removed from the concept description because of some past (and forgotten) training example. These two kinds of information—what is provable of the remembered past positive example and the history of particular features—constrain the revisions that the inductive operators can make to update the concept description.

3.2. Conserving information about features

How LAIR processes training examples is also slightly different from most incremental learning approaches, which typically leave the current concept description unchanged if it correctly covers a positive training example (or correctly rejects a negative one). In contrast, LAIR takes each example as a source of information about the desired concept description and reformulates the concept description so that it accounts for as much of an example description as possible. For each feature dropped from a concept description in a generalization step and for each feature mentioned in an example that is missing from the concept

description, LAIR tries to add a related feature to the concept description. If the inductive operators can add these related features to the concept, then some of the information about the example is conserved in the concept description.

As an example, suppose that LAIR's current concept hypothesis for the concept *chair* is *(has-part object-1 legs) & (has-part object-1 seat) & (sit-on people object-1)*. If a new positive example did not mention *(has-part object-1 legs)*, then this feature must be dropped from the concept description as a required feature. However, if the domain theory had the inference rule *(has-part =object legs)* → *(stable =object)*, then *(stable =object)* might be a valid addition to the concept description, if it is provable of the current positive instance. Similarly, if the new positive example had the feature *(has-part object-1 cushions)* and the domain theory specifies *(has-part =object cushions)* → *(soft =object)*, then LAIR attempts to add *(soft object-1)*, or some feature that it might imply, back into the concept description. Whether this type of modification actually occurs is controlled by the preconditions of the inductive operators, which we outline below. In general, this approach aims to reconcile the apparent differences between the current concept definition and the current example rather than simply checking to see if the concept definition correctly accepts or rejects the example.

3.3. Using deduction to classify and modify example descriptions

Deduction underlies LAIR's ability to classify instances as positive or negative instances of a concept, and this process also supports the inductive operators that reformulate the current concept description during learning. Deduction is done through a backward-chaining theorem prover with negation by failure (cf. Sterling & Shapiro, 1986).

If LAIR can derive the current concept description from the domain theory and the example description, then the example is classified as a positive instance of the concept. Otherwise, it is classified as a negative instance. Specifically, the domain theory and any positive example of some concept entail a correct description of that concept: if LAIR cannot prove a concept description from an example and the domain theory, then that example must be a negative example of the concept. This is consistent with negation by failure, and also allows us to omit negated features from example descriptions rather than representing them explicitly.

LAIR relies on deduction throughout the learning process. First, the classification process is invoked to determine whether the current concept description is consistent with a new training example. Second, LAIR uses deduction to conserve information about features that are dropped from the concept description and about features that appear in a positive example but do not appear in the concept description. Deduction is also used to derive features for specializing a concept description, so that it rejects a negative example. LAIR implements these generalization and specialization processes in two steps. The first step deductively extends the example description to include the new features. The second step is controlled by the inductive operators, described in the next section, and involves modifying the concept description to include these features as new required or forbidden features. Together, these steps define the process of constructive induction.

3.4. Inductive operators and constraints on concept revision

Table 1 gives the four inductive operators that LAIR uses to revise its concept description during learning: a feature can either be added to or dropped from the description as either a required feature or as a forbidden feature. Note that the operators' preconditions reference both the current example description and the remembered past positive example (but there is nothing that limits the remembered set size to two examples).

The inductive operators' preconditions specify whether the feature must be true or not true of the current example and the remembered positive example. The preconditions also constrain a feature to have a particular status with respect to the current learning task. A feature may have a status tag whose value can be (a) *required*, if the feature is currently a required feature in the concept description, (b) *forbidden*, if the feature is currently a forbidden feature, (c) *can't-forbid*, if the feature was true of some previously-seen positive example and hence cannot be added as a forbidden feature, or (d) *can't-require*, if the feature had once been a required feature of the concept description but was later removed during a concept revision step. This status information is stored with the feature's representation in the domain theory. A feature will have no status tags until it has been seen in some example description.

Table 1. Inductive operators.

Add-required Operator: Adds an unnegated (required) feature R to the concept description.		
If		R does not have status tag *can't-require* or *required*
	&	R can be proven true of the remembered past positive example(s)
	&	R cannot be proven true of the current example if it is negative
Then		add R to the concept description and set its status tag to *required* and *can't-forbid*
Add-forbidden Operator: Adds a negated (forbidden) feature F to the concept description.		
If		F does not have status tag *can't-forbid* or *forbidden*
	&	F cannot be proven true of the remembered positive example(s)
	&	F can be proven true of the current example if it is negative
Then		add F to the concept description and set its status tag to *forbidden* and *can't-require*
Drop-required Operator: Drops an unnegated (required) feature R from the concept description.		
If		R does not have status *drop-required*
	&	R cannot be proven true of the remembered positive example(s)
Then		drop F from the concept description and remove *required* from its status tag and add *can't-require* to its status tag
Drop-forbidden Operator: Drops a negated (forbidden) feature F from the concept description.		
If		F does not have status *can't-forbid*
	&	F can be proven true of any remembered positive example
Then		drop-F from the concept description and remove *forbidden* from its status tag and add *can't-forbid* to its status tag

The preconditions of the inductive operators in Table 1 are quite intuitive. For example, the *add-required* operator adds a feature to the concept description only if the feature is provable of the current positive example (or not provable, if the current example is negative), provable of the remembered past positive example, and has never been dropped from the concept description in the past. The last constraint is important. If a feature P has the status *can't-require*, then that means it had been dropped from the concept description by the *drop-required* operator. This would only have happened if LAIR could not prove P to be true of some previously-presented (now forgotten) positive example. Clearly, P must not be re-added at some future point as a required feature. Thus, LAIR does not need to remember all positive or negative examples, only its decisions about features appearing in those examples during its concept revision processes.

The other inductive rule that is used to specialize a concept description is the *add-forbidden* operator. The constraints on the feature in this operator's preconditions ensure that a feature will not be added as forbidden if it has ever been known to hold for some past positive example (i.e., has the status tag *can't-forbid*). The rationale for this constraint is also intuitive. If a feature that was true of a previous positive example were added as a forbidden feature, then the correct concept description could not be derived from the domain theory and that previous positive example. More to the point, it would misclassify this previously-seen positive example.

The preconditions of the operators imply that at most three inductive operators can be applied to a given feature: either drop-forbidden–add-required–drop-required, or drop-required–add-forbidden–drop-forbidden.

4. Control strategy for induction

In this section, we outline LAIR's control strategy for learning. The task of learning a new concept begins with the presentation of a positive example. The concept description is initialized to the description of this example, and it will be remembered throughout the learning task. LAIR is then shown new examples; the example which may be either positive or negative. Thus, at any time LAIR is considering three structures: the description for the remembered past positive example, the current concept description, and the description for the current example.

Tables 2a and 2b give LAIR's algorithm for updating the concept hypothesis when a positive example is presented and when a negative example is presented. We elaborate each step of this algorithm in the next sections.

4.1. Processing positive examples

When LAIR is given a positive example, it first sets the status tag of each feature appearing in the example to *can't-forbid*. Recall that this marker is stored on each feature in the knowledge base and serves as LAIR's memory that these features appeared in some positive example, which itself will be forgotten.

Table 2a. Algorithm for processing positive instances.

If the current example E is positive

1. For each feature D in E
   1.1 Apply the Drop-forbidden operator to D.
   1.2 If D is not tagged *required* or *can't-require*, then apply the Add-required operator to D. If unsuccessful, save D with a pointer to E for information conservation.
2. For each required R in the concept description, apply the Drop-required operator to R. If successful, save R with a pointer to the remembered past positive example for information conservation.
3. For each forbidden F in the concept description, apply the Drop-forbidden operator to F.
4. For each feature D and its corresponding example X (from steps 1.2 and 2) saved for information conservation,
   4.1 Initialize a required stack to contain generalizations of D.
   4.2 Repeat
      4.2.1 Take a feature G from the required stack.
      4.2.2 If G has already been examined, or is tagged *required*, or *can't-require*, go to 4.2.
      4.2.3 Apply Add-required operator to G. Exit loop if successful.
      4.2.4 Apply Drop-required operator to G.
      4.2.5 If G is true of X, add generalizations of G to required stack.
      Until step 4.2.3 is successful or the required stack is empty.

Table 2b. Algorithm for processing negative instances.

If the current example E is a negative example

1. Try to derive each of the required and forbidden features in the concept description from E. If a required cannot be derived, or a forbidden can be derived, go to step 4.
2. Set a required stack to all features that appear in the remembered past positive example and are not derivable from E. Set a forbidden stack to all features that appear in E and are not derivable from the past positive example.
3. While both the required and forbidden stacks are not empty do
   3.1 If the required stack is not empty,
      3.1.1 Remove a feature R from the required stack.
      3.1.2 If R has already been checked, then go to step 3.
      3.1.3 Apply Add-required operator to R. If successful, go to step 4.
      3.1.4 Apply Drop-required operator to R.
      3.1.5 Add all R's generalizations to the required stack and all $-$R's generalizations to the forbidden stack iff R is true of the remembered past positive example and not true of E. Go to step 3.
   3.2 If the required stack is empty and the forbidden stack is not empty
      3.2.1 Remove a feature F from the forbidden stack.
      3.2.2 If F has already been tried, then go to step 3.
      3.2.3 Apply Add-forbidden operator to F. If successful, go to step 4.
      3.2.4 Apply Drop-forbidden operator to F.
      3.2.5 Add all F's generalizations to the forbidden stack and all $-$F's generalizations to the required stack iff F is true of E and not true of the past positive example. Go to step 4.
4. If the concept description rejects E, then report success. Otherwise, report failure.

LAIR next tries to apply the add-required operator to each feature in the current example that it has not seen before (step 1.2 in Table 2a). This tests that each of these features is also true of the remembered past positive example, either through a direct match or some derived match (see the preconditions of the add-required operator in Table 1), and does not have the status *can't-require*. Any feature that does not meet the add-required operator's preconditions has its status tag changed to *can't-require*, meaning it was not provable of either the current positive or the remembered positive example. This will ensure it is never subsequently added back as a required feature. These *can't-require* features, along with a tag indicating the example that they came from, are saved for information conservation.

The required features in the concept description are then tested to see if they are true of the current example, by applying the drop-required operator. To conserve information, the system saves any required feature that it drops, with a tag indicating it came from the concept description. At this point, LAIR has formed a new set of required features from two sources: never-seen-before features appearing in the current positive example (step 1) and the old set of required features in the concept description (step 2).

The last step in making a concept description consistent with the current positive example involves checking that the forbidden features are not true of the current positive example (step 3). This is done using the drop-forbidden operator, which removes any forbidden feature from the concept description that is true of the current positive example. At this point, LAIR has a concept description that has been revised to cover the current positive example. All the required features and forbidden features have been checked and status tags of features have been updated by inductive operators.

Step 4 in Table 2a corresponds to the information conservation process. LAIR keeps track of all the features it marked as dropped for information conservation. These features were required features that could not be proven true of the current positive example and features of the current positive example that could not be added to the concept description as required. The rationale behind information conservation is this: (a) features that are true of a positive example (but not true of all positive cases) may still reflect some important information about the concept description, simply by their very presence in the positive example, and (b) even though these features failed to meet the conditions for required features, perhaps some feature they imply is an essential part of the concept description. LAIR tries to conserve some of the import of these features, rather than dismissing them as irrelevant, by attempting to prove some other feature they imply would be a valid addition to the concept description.

We will now step through the process of conserving information about dropped features. The first step involves retrieving all inference rules from the domain theory in which a dropped feature P appears as an unnegated antecedent. We refer to the consequents of these retrieved rules as *generalizations* of P. If the domain theory contained the inference rules $P \ \& \ O \rightarrow R$ and $P \ \& \ S \ \& \ T \rightarrow U$, then R and U would be generalizations of P. These are simply the features that P, along with other features, implies.

Next, all of the generalizations of a dropped feature are placed on a *required stack*, and tagged with the information that indicates they come from the current example. LAIR tries to apply the add-required operator to the items on this stack, which triggers a proof procedure. Using the above example, LAIR would select the first item on the required stack, say R. If R does not match the add-required operator's preconditions, then LAIR searches

for and places *its* generalizations on the top of the required stack as new candidates as long as the following constraint is satisfied: R must be true of at least the remembered past positive example or the current positive example (step 4.2.5). The rationale for this constraint is that LAIR is using R to search for features that may be true of *both* examples (to meet the add-required operator's preconditions). Therefore, it makes sense to insist that R is true for at least one of them before expending deductive effort on its generalizations. The process of trying to prove that a required-stack feature meets the add-required operator's preconditions, and retrieving more generalizations if it does not, continues until LAIR succeeds with one feature (in which case LAIR has conserved information about the dropped feature, and stops) or until the required stack becomes empty.

This ends LAIR's two-stage processing of positive examples: make the current concept description correct with respect to the current positive example, and try to conserve information about any feature, from both the concept description hypothesis or from the current positive example, that could not be directly retained or included as a required feature.

4.2. Processing negative examples

When presented with a negative example, LAIR first determines whether the current hypothesis rejects the negative example (step 1 in Table 2b). It does this by checking for a match (direct or derived) between the required and forbidden features of the concept hypothesis and the example description. If the concept description does not reject the negative example, then LAIR attempts to specialize it. Like Drastal, Meunier, and Raatz's (1989) MIRO, our system tries to specialize a concept description by adding required features rather than forbidden features. This results in more concise concept descriptions for our domain, as required features typically rule out more negative examples than forbidden features.

The specialization effort proceeds in a manner similar to that used for information conservation. LAIR uses information about the remembered past positive example and the current negative example to search for features that might serve as new required or forbidden features. Search stops when a discriminating feature can be added to the concept description. A discriminating feature is defined to be a feature that is true of one example (either the current negative or the remembered past positive) and not true of the other example.

LAIR initializes its required stack to all the discriminating features that occur in the remembered past positive example. The required stack represents a (changing) set of candidates for a new required feature to specialize the concept. LAIR also initializes a *forbidden stack* to all the discriminating features that occur in the current negative example. The forbidden stack represents a (changing) set of candidates for new forbiddens. Although every member of the stack is a discriminating feature, they still may not be valid additions to the concept description depending on their status tags. Specifically, the discriminating features that are required candidates may have the status tag *can't-require*, indicating that they were dropped from the concept description on some previous revision step. The discriminating features on the forbidden stack may have a tag of *can't-forbid*, indicating they had appeared in some previous positive example.

LAIR selects the first feature, R, from the required stack and checks whether the preconditions of the add-required operator are satisfied (step 3.1.3). If they are, and the resulting

concept description would correctly reject the negative example, then the specialization task is done. If the feature R cannot be added as a required feature, then the drop-required operator is applied. Next, if R is a discriminating feature (true of one example, but not the other), its generalizations are retrieved and added to required stack (step 3.1.5). This search for new, more general features that might be valid additions as required features is completely analogous to information conservation. However, LAIR can also use a feature R from the required stack to search for possible forbidden candidates. If R has failed as a required feature, then a generalization of −R might be a forbidden feature of the concept. The rationale is this: if R is true of the remembered past postive example and not true of the current negative example, then −R is not true of the remembered past positive example and true of the current example. Thus, −R is simply another form of the discriminatory feature R, so its generalizations may also be discriminating features. Therefore, LAIR finds rules such as $S \,\&\, -R \rightarrow T$, and augments the forbidden stack with T.

If LAIR tries but fails to add a required feature to the concept description, it updates the required and forbidden stacks in the above manner, and then switches its focus to the first feature on the forbidden stack. If this feature F meets the preconditions of the add-forbidden operator, then the specialization process ends. If it does not, then the drop-forbidden operator is applied. Next, if F is a discriminating feature, then F's generalizations are added to the forbidden stack and −F's generalizations are added to the required stack. This is the mirror image of the process described in the previous paragraph. For example, if the domain theory contained the rules $U \,\&\, F \rightarrow V$ and $W \,\&\, -F \rightarrow X$, then V would be added to the forbidden stack and X would be added to the required stack.

LAIR keeps trying to add a new required or forbidden feature to the concept description (using the forbidden stack only if the required stack is empty), until the concept description correctly excludes the negative case or until both the required and forbidden stacks are empty.

4.3. Observations on the learning strategy

The main objective of the above strategy is to find a concept description that is consistent with the remembered examples. An important constraint on processing negative examples is that LAIR only uses discriminating features to search for additional required and forbidden features. However, this constraint is sound only if it does not prevent LAIR from finding a discriminating feature that can specialize the description to exclude the current negative example. We will show that this restriction is sound for a discriminating required feature; the case for a forbidden feature is completely analogous.

By definition, a discriminating required feature is provable of the remembered past positive example and unprovable of the negative example. Each possible proof of the required feature from the positive example can be viewed as an AND-tree. The root node is the required feature and the leaf nodes are the features in the positive example. Since the tree represents the proof of the required feature from the positive example, every node in the tree must be derivable from the positive example, or else the proof would fail. Because the root node corresponds to the discriminating required feature, by definition it is unprovable of the negative example. Therefore, at least one of the child nodes of the root must also be unprovable of the negative example. This argument can be applied recursively, so there must

be at least one path from the root to the leaf nodes consisting of discriminating features. The root node is thus reachable from the leaf nodes by a path of discriminating features.

The learning strategy also has a secondary goal of trying to incorporate as much information as possible from the current positive example into the concept description, using the inductive operators. The concept description often becomes more detailed, in the sense of including more features, even when it is consistent with the current example. However, as some of the experimental results we present below indicate, the extra inferencing work of information conservation allows LAIR to reach a learning criterion with fewer examples.

5. An example of acquiring and using a domain theory

In this section, we present several learning runs from a session in which LAIR acquired a small domain theory for the concept *cup*. These runs illustrate details of the control strategy and also demonstrate how LAIR both constructs a domain theory through learning from examples and then uses this domain theory to learn subsequent concepts. Table 3 presents the example descriptions for the learning tasks we discuss below.

A teacher interacts with LAIR on three occasions: to indicate that a new sequence of examples for a new concept is starting, to present an example, and to inform LAIR that it is done learning a concept description, at which point its definition is added to the domain theory. This last action clears all status tags on features that have accumulated during the learning task.

Table 3. Examples for the cup domain.

Name	Description
uninsulated object	(body obj body-obj)(light obj)(small body-obj)(cylindrical body-obj obj) (contents obj contents-obj)(covered obj)(hot contents-obj)
insulated object	(body obj body-obj)(light obj)(small body-obj)(cylindrical body-obj) (contents obj contents-obj)(covered obj)(hot contents-obj)(insulated obj)
soup bowl	(concavity obj concavity-obj)(upwards concavity-obj)(useful obj) (contents obj contents-obj)(body obj body-obj)
brick	(bottom obj bottom-obj)(hard obj)(red obj)(flat bottom-obj)
empty cup	(body obj body-obj)(small body-obj)(cylindrical body-obj)(bottom obj bottom-obj) (flat bottom-obj)(handle obj)(concavity obj concavity-obj)(upwards concavity-obj) (light obj)
gyroscope	(rotates obj)(balanced obj)
balanced cup	(contents obj contents-obj)(balanced obj)(body obj body-obj)(handle obj)(light obj)
uncovered pot	(handle obj)(contents obj contents-obj)(stew contents-obj)
covered pot	(handle obj)(contents obj contents-obj)(stew contents-obj)(covered obj)
flashlight	(body obj body-obj)(light obj)(small body-obj)(cylindrical body-obj)
suitcase	(body obj body-obj)(light obj)(handle obj)(bottom obj bottom-obj) (flat bottom-obj)(portable obj)(contains-clothes obj)
purse	(clasp obj)(leather obj)(contents obj contents-obj)(personal contents-obj)(handle obj)
lift scenario	(was-lifted obj)
tea cup	(bottom obj bottom-obj)(flat bottom-obj)(concavity obj concavity-obj) (cylindrical body-obj)(small body-obj)(body obj body-obj)(was-lifted obj) (upwards concavity-obj)

Table 4. Task sequence for acquiring the domain theory for cup.[a]

Task	Example Sequence	Learned Concept Description
1	+uninsulated object +soup bowl −insulated object	(hot =contents-obj)&(contents obj =contents-obj) & (body =body-obj) & not(insulated =obj) → (hot =body-obj)
2	+brick +empty cup	(bottom obj =bottom-obj) & (flat =bottom-obj) →(stable =obj)
3	+gyroscope +balanced cup	(balanced =obj) → (stable =obj)
4	+soup bowl +empty cup	(body obj =body-obj) & (concavity =obj =concavity-obj) & (upwards =concavity-obj) → (open-vessel =obj)
5	+soup bowl +open pot −covered pot	(contents obj =contents-obj) & not(covered =obj) → (open-vessel =obj)
6	+uninsulated object +flashlight −uninsulated object	(cylindrical =body-obj) & (small =body-obj) & (light =obj) & (body =obj =body-obj) & not(hot =body-obj) → (graspable =obj)
7	+suitcase +purse	(handle =obj) → (graspable =obj)
8	+flashlight +insulated object −uninsulated object +balanced cup	(graspable =obj) & (light =obj) & (body =obj =body-obj) → (liftable =obj)
9	+lift-scenario +tea cup	(was-lifted =obj) → (liftable =obj)
10	+tea cup +empty cup +balanced cup	(open-vessel =obj) & (stable =obj) & (liftable =obj) & (body =obj =body-obj) → (cup =obj)

[a] +means positive example, −means negative example

Table 4 lists the order of tasks used in these runs to acquire the basic domain theory. For each task in the table, we have listed the example sequence and the resulting concept definition that is added to the domain theory. A plus sign before an example indicates it is a positive instance; a minus sign indicates it is a negative instance.

When LAIR begins, there are no features, rules, or concept definitions in the domain theory. Features are added to the general knowledge base by virtue of their appearance in example descriptions. The domain theory grows as LAIR progresses through the learning tasks. We will not elaborate each step of each task, but discuss portions of some tasks that illustrate the inductive processes described earlier.

5.1. Acquiring initial domain rules

In Task 1, the concept description is initialized to all features of the first positive instance, *+uninsulated object*. LAIR tries to prove that all these required features are true of the next positive instance, *+soup bowl*, and that all of soup bowl's features are true of the

remembered positive case. This is not possible for many features, such as *useful* or *cylindrical*, and there is no domain theory at this point that LAIR could use to conserve information about them. Thus, the resulting concept description after +soup bowl is *(hot contents-object) & (contents-object contents-object) & (body object body-object)*.

For the third example, *−insulated object*, the feature *(insulated object)* meets the constraints of the add-forbidden operator and can be added to specialize the concept description to exclude this negative case. At this point, LAIR was informed the concept *(hot =body-object)* was learned, and the system added its first rule to the domain theory.

The goal of the second learning task was to acquire a definition for *stable*. There are several differences between the two positive training examples, *+brick* and *+empty cup*. All these features are dropped as required features, leaving only *(bottom object bottom-object) & (flat bottom-object)* as the concept description. Of all the dropped required features, only *(body object body-object)* had a generalization in the domain theory (*(hot =body-object)* from Task 1). However, LAIR's attempt to conserve information about body-object using this generalization fails, because *(hot =body-object)* cannot be proven true of both the current example and the remembered past positive example. The second definition added to the domain theory is *(bottom =object =bottom-object) & (flat =bottom-object) → (stable =object)*.

5.2. Specializing the concept hypothesis

Tasks 6 and 8 illustrate how LAIR uses generalizations to specialize a description to exclude a negative case. We will step through Task 8, because the domain theory is more developed at this stage and provides a better illustration of how LAIR searches the domain theory for specialization candidates.

In Task 8, none of the features that were unique to the first two positive examples, *+flashlight* and *+insulated-object*, can be conserved with higher-level features, so the concept description after the second example is *(cylindrical body-object) & (small body-object) & (light object) & (body object body-object)*. When *−uninsulated object* is presented next, the current concept description is too general to reject it. LAIR cannot add features of the negative example as forbiddens because none of them match the add-forbidden operator's preconditions. Therefore, the system must search for a specialization. The required-stack is initially empty (there is no feature true of the past positive and not true of the current negative example) and the forbidden-stack is initialized with features of the negative example description, *(contents object contents-object) & (covered object) & (hot contents-object)*.

The first feature on the forbidden-stack, *(contents object contents-object)*, has a status tag of *can't-forbid*, so it fails to match the add-forbidden operator's preconditions. Therefore, LAIR searches for and finds generalizations of *(contents object contents-object)* for the forbidden stack: *(hot =body-object)* and *(open-vessel =object)*. LAIR also searches for generalizations of *−(contents object contents-object)*, but this feature does not appear in any rule definition, so there is nothing new to add to the required stack at this point.

Because the required stack is empty, LAIR tries a candidate from the forbidden stack again. It successfully applies the add-forbidden operator to *(hot body-object)*. The specialization task ends, leaving the concept hypothesis as *(cylindrical body-object) & (small body-object) & (light object) & (body object body-object) & −(hot body-object)*.

When *+balanced cup* is presented as the fourth example, several features must be dropped from the concept hypothesis. LAIR conserves information about one of them, *(small body-object)*, by proving its generalization, *(graspable object)*, true of both the remembered past positive example and the current positive example. The final concept description is *(graspable =object) & (light =object) & (body =object =body-object) → (liftable =object)*.

5.3. Acquiring the cup concept with information conservation

Using the domain theory acquired in Tasks 1-9, LAIR was able to acquire the concept cup from only three positive examples in Task 10. This task provides a good illustration of the search for generalized features as part of information conservation, so we will step through the process to learn this concept.

When LAIR is given the second positive instance in this sequence, *+empty-cup*, it cannot include its feature *(handle object)* in the concept description because this feature is not provable of the remembered past positive example. However, LAIR is able to conserve information about this feature by adding *(graspable object)* to the concept hypothesis. This is accomplished by using different parts of the domain theory. *+empty-cup* is graspable because it had a handle (learned in Task 7) and the remembered positive example, *+tea cup*, is graspable according to the definition learned in Task 6.

Given the third example, *+balanced-cup*, LAIR must drop the required features *(bottom object bottom-object)* and *(flat bottom-object)* because these are missing from this example. It conserves information about these features by adding *(stable object)* to the concept hypothesis: this feature is provable for both the current example and the remembered past positive example. In addition, LAIR can conserve information about one of *+balanced-cup's* own features, *(contents object contents-object)*, by adding its generalization, *(open-vessel object)* to the concept hypothesis. The remembered past positive example had several features that implied that it too was an open-vessel, according to a different definition. The final description for cup that emerged with this example sequence is *(open-vessel =object) & (stable =object) & (liftable =object) & (body =object =body-object) → (cup =object)*.

5.4. Observations on information conservation

As the above tasks demonstrate, information conservation plays an important role in LAIR's learning. It is interesting to note that this process makes LAIR's generalized descriptions equivalent to those generated by Buntine's (1988) generalized subsumption method for organizing generalizations in a knowledge base. Using Buntine's (1988) example, suppose that *(small object) & (fluffy object) & (dog object)* is LAIR's remembered past positive example for the concept *cuddly-pet*. The system makes this description its first concept

hypothesis. If the next positive instance is *(fluffy object) & (cat object)*, LAIR would drop the features *(small object)* and *(dog object)* from the concept description, and then try to conserve information about these two features along with example 2's feature *(cat object)*. Suppose that the domain theory contained the rule definitions *(cat =object)* → *(pet =object)*, *(cat =object)* → *(small =object)*, *(dog =object)* → *(pet =object)*, and *(pet =object)* → *(tame =object)*. LAIR would not find any rules in the domain theory in which *(small object)* or *(fluffy object)* appears as an antecedent. The feature *(dog object)* has one generalization, *(pet object)*, so LAIR would try to prove *(pet object)* is true of the remembered positive instance and the current one. It can, so it adds *(pet object)* to the concept description. The feature *(cat object)* has generalizations *(pet object)* and *(small object)*; *(pet object)* is already a required feature of the concept description but *(small object)* can be proven true of the remembered past positive instance as well as the current instance, so it is re-added to the concept description. This yields *(small object) & (fluffy object) & (pet object)*, which is the same description produced by Buntine's generalized subsumption. The information conservation process is responsible for the equivalence, because it generalizes two forms according to deductive relations among related features, rather than according to syntactic overlap. Generalized subsumption can be viewed in the same way.

6. An experimental framework for evaluating LAIR

In this section, we describe the framework we developed for empirically evaluating LAIR and the dependent and independent variables that we used. Our purpose in performing an empirical evaluation of LAIR was twofold. First, we wanted to evaluate the advantage of its controlled constructive induction and information conservation approach over an instantiation of "uncontrolled" constructive induction. By uncontrolled constructive induction, we mean a process that extends an example description with all possible features that are derivable from the example description and the domain theory, in a manner similar to that used in INDUCE (Larson & Michalski, 1977). Second, we wanted to determine how LAIR's inductive and deductive capabilities were affected by different *learning scenarios*. A learning scenario consisted of three major components: (a) an automatically constructed knowledge-base of concepts and rule definitions, which constituted a domain theory, (b) a target concept and definition randomly selected from a knowledge base, and (c) a set of automatically-generated training examples. Learning scenarios differed in the global characteristics of these components, such as the size of the domain theory, sequencing of positive and negative examples, or the degree of irrelevant features in the example descriptions. These variations, along with some manipulations of LAIR's deductive components, were our independent variables. We did multiple experimental runs within a scenario to get a stable estimate of two dependent measures: the number of examples needed to reach a pre-defined learning criterion and the number of inferences made to reach this criterion.

Table 5. Experimental parameters and default settings.[a]

Parameter Type and Name	Default Settings
Knowledge Base Characteristics	
Number of Levels	5
*Concepts per Level	30
Number of Rule Definitions per Concept	3
Number of Antecedents per Rule Definition	5
Pos. & Neg. Antecedents per Rule Definition	p = 0.6 of generating a positive antecedent
Distance of Antecedents from Level N Rule Definition Consequent	one antecedent from Level N-1; For each remaining antecedent, p = 0.6 to be from Level N-1, p = 0.4 to be Level 0
Training Sequence Characteristics	
*Distribution of Pos. & Neg. Examples	random
*Padding Factor for Adding Irrelevant Features per Example	1.2
Learning Strategy Characteristics	
*Information Conservation	see text
Level of Target Concept	2, 3, 4, and 5

[a]Starred entries were also manipulated as independent variables.

Table 5 summarizes the important parameters and their default values that defined features of the knowledge base, the target concept, and the training sequence. The next sections explain each of these three learning scenario components in turn. Following that, we present our implementation of an uncontrolled constructive induction system, which served as a baseline against which to compare LAIR's performance. Finally, we discuss the independent and dependent variables in more detail.

6.1. Constructing and varying features of the knowledge base

A knowledge base is a set of randomly-named concepts with randomly-generated rule definitions, defined by the following parameters.

Number of levels. Each knowledge base has primitive features that have no associated rule definitions. These features comprise what we call level 0 in the knowledge base. Level 1 features use only level 0 features in their rule definitions, level 2 features use level 1 and level 0 features in their rule definitions, and so on. An N-level knowledge base has N such levels. The number of levels in a knowledge base is one global characteristic that partially defines the inferential distance between concepts that participate in the definition of other concepts. As Table 5 indicates, all the knowledge bases used in our experiments had five levels.

Number of concepts populating each level. Each knowledge base has a certain number of concepts at each level. This number can be specified independently for each level, so that a knowledge base can be given a particular shape (e.g., a broad base consisting of many concepts and a narrow top with few concepts).

Number of rule definitions per concept. This parameter corresponds to the number of alternative rule definitions any concept may have. For example, in the cup domain there were two alternative definitions for graspable. As Table 5 shows, we used three definitions for all concepts in our knowledge bases.

Number of antecedents per rule definition. This parameter specifies the number of antecedents for every concept definition in the knowledge base.

Distribution of positive and negative antecedents. This parameter defines the probability that any particular antecedent in a rule definition will be positive or negative. With the default value given in Table 5, most of the rule definitions typically had 3 or 4 positive antecedents and 1 or 2 negated antecedents.

Antecedent distance. This parameter controls the levels from which antecedents are randomly selected to automatically construct a concept definition. For example, the default setting for this parameter specifies that each antecedent for a level-N concept has a 0.6 probability of being drawn from level N-1 and a 0.4 probability of being drawn from the level-0 primitive features. The value of this parameter is constant for all concept definitions in the knowledge bases and was fixed for all experiments at the value given in Table 5.

6.2. Selecting the target concept and generating examples

The *target concept* is the concept given to LAIR for learning. It is randomly selected from a particular level in the knowledge base and referred to by its level, as in "a level-3 target concept." We randomly choose one of the target concept's three rule definitions as the definition to be acquired. Because LAIR uses the knowledge base to learn the definition, this definition is temporarily removed from the knowledge base during the learning task.

All examples use only level-0 primitive features. Positive examples of a target concept are generated by back-chaining from the concept's rule definition to level-0 features. Negative examples are created by first generating a positive example and then randomly doing perturbations (adding a feature as a forbidden or dropping a required) until the example is negative. Then, a second parameter controls how many additional perturbations of this new negative example are done, and hence how much of a near- or far-miss the resulting negative example will be.[2]

Another parameter, which we call the *padding factor*, controlled how many irrelevant features might also be included with any example. The default setting for this parameter was 1.2. The padding factor defines a percentage of the original relevant features that is used to set the number of additional irrelevant features. For example, 100 relevant example features $\times$ 1.2 padding factor = 120, and 120 – 100 original features = 20 irrelevant features to be added. This can be expressed as a ratio of irrelevant to relevant features, e.g., 20/100 = 0.2. With a default padding factor of 1.2, the size of example descriptions typically ranged from 15—40 features for a level-5 target concept.

6.3. A deductive closure version of LAIR

For purposes of having a baseline for evaluating LAIR's performance, we implemented an "uncontrolled" constructive induction system. There can be many alternative implementations of the idea of constructive induction, which itself does not specify any particular

learning algorithm. Our operational definition of uncontrolled constructive induction was complete deductive closure on a training example and the knowledge base. We implemented this as an alternative version of LAIR, which we call the *closure system*. The closure system is a version of LAIR with a preprocessor that replaces the description of an example with its deductive closure. Both LAIR and the closure system learned the same target concepts. For each target concept, the examples that LAIR used to reach criterion were given to the closure system. The closure system might reach criterion sooner and not use all the examples that LAIR needed, or it could require additional examples. The important point is that both the concepts to be learned and the training sequences were held constant for both systems.

It is important to reiterate how the closure system differs from LAIR's approach. First, LAIR uses deduction to reconcile the current concept hypothesis with the current example, dropping required features or adding negated features as necessary. It then does additional deduction to conserve information about features dropped from the current example and the current hypothesis. There are two constraints that set an upper bound on the deductive effort to include new features from the knowledge base to achieve information conservation or specialization: (a) LAIR stops information conservation or specialization as soon as it has incorporated one generalization, not all, of a feature, and (b) its recursive effort to retrieve and prove a generalization of some feature continues only if it makes heuristic sense to do so.[3] These constraints are unnecessary in the closure version of LAIR, since no new features can be derived from the examples. Every feature that can be derived from the initial example description and the knowledge base is included in the initial example description, and is considered for inclusion in the concept hypothesis.

The closure system is closely related to the classical learning algorithm for conjunctive descriptions, described in Kearns et al. (1987). The closure system generalizes and specializes unnegated features exactly as in the classical learning algorithm. However, the closure system specializes with negated features only when there is no other method of making current concept description consistent with a negative example. As mentioned earlier, we have empirically found that this modification is preferable in our test domains. To our knowledge, the classical learning algorithm is the most efficient algorithm for learning conjunctive propositional descriptions. The performance of the closure system, reported in the experiments below, should be interpreted in light of these features of its implementation.

6.4. Independent variables

This section outlines the independent variables we manipulated and our rationale for focusing on these particular aspects for an empirical evaluation. They concern features of the learning scenario, the training examples, and the training example presentation order.

Irrelevant Example Features. A constructive induction system derives new features from example descriptions and the domain theory. It is important to assess the sensitivity of the method to different relative amounts of relevant and irrelevant features appearing in example descriptions.

Information Conservation. We have argued that information conservation is a potentially powerful aspect of our approach, and yet clearly it involves what might seem like unnecessary

inferential overhead: LAIR spends time semantically reconciling the current concept description with the current example even if the concept description syntactically matches all the example's features. We examined several information conservation methods that differed in their degree of deductive effort and contrasted them with a baseline case of no conservation. We were particularly interested in assessing whether information conservation lets LAIR learn a target concept with fewer examples.

Positive and Negative Example Bias. Pilot experiments in the cup domain indicated that different orders and frequencies of positive and negative examples influence how quickly LAIR learns a target concept. It is important to understand how a particular learning algorithm is affected by these factors, because there are learning situations in which one can control example type and order, and other situations in which they may be biased (e.g., one type may be more likely to occur in the environment). Therefore, we wanted to explore how constructive induction was affected by different degrees of positive- and negative-example bias in the training sequence.

Knowledge-base width. Given the way we construct our concept definitions, the number of concepts at each level—knowledge-base width—influences the degree of concept definition overlap. The three definitions for each concept at level N are constrained to have a certain percentage of their antecedents from level N-1 and level 0. The fewer concepts that populate each level, the more overlap there will be in those three definitions. In addition, all examples are created from the primitive features at level 0. If this level is very narrow, there will also be a high degree of similarity among positive and negative examples for any particular concept. These factors can be viewed as defining the density of features in the concept search space and we wanted to assess its impact on the deductive and inductive components of constructive induction.

Knowledge-base shape. By specifying different numbers of concepts at each level, it is possible to construct knowledge bases of different shapes. For example, a square shape can be formed by having equal numbers of concepts at each level, while a pyramid-shape can be created by having a large number of level-0 concepts and systematically decreasing the number of concepts at each successive level. Like knowledge-base width, knowledge-base shape defines general characteristics of the search space. It changes the density of features and the overlap among concept definitions as a function of distance from the level-0 primitive features. For example, the wide base of a pyramid shape decreases the likelihood of overlap among the training examples constructed for level-5 target concept definitions, which themselves are constrained by the pyramid's narrowing top to have a high degree of overlap with other level-5 definitions. We wanted to determine how these other search space characteristics affected the search for features to generalize and specialize a concept hypothesis.

6.5. Dependent variables: Trials to criterion and inferencing effort

We measured two dependent variables. The first was the number of examples LAIR and the closure system required to learn to criterion. We tested both systems after every second training example, using an automatically-generated set of 20 positive and 20 negative test examples. Learning to criterion was defined to be 95% accurate on classifying this test set.

The second dependent variable measured deductive effort. We counted the number of inferences LAIR and the closure system each made while learning to criterion. We then computed the ratio of LAIR's inferences to the total number of inferences made by both systems. This ratio provides a concise assessment of LAIR's deductive effort relative to the closure system's effort. When both systems make the same number of inferences, the value of the ratio is 0.5. A value less than 0.5 means LAIR made fewer inferences than the closure system and a value greater than 0.5 means it made more inferences.

By measuring both examples to criterion and the number of inferences, we can evaluate LAIR's behavior from both an inductive and deductive perspective. In addition, we can assess whether there is a trade-off between examples needed and inferential effort under particular learning scenarios.

7. Experimental designs and results

We did not factorially cross all possible levels of each independent variable. This would have yielded an unwieldy factorial design and any resulting interactions would have probably been uninterpretable. Instead, we crossed target-concept level with each of the other independent variables, because the other variables may interact with the inferential distance of the target concept from the example descriptions.

We performed 6 different runs per variable crossing to obtain stable estimates of trials to criterion and inference ratios. Thus, a three-level variable crossed with four different target concept levels yields 12 different combinations, run six times each for a total of 78 independent runs. The trials-to-criterion and the number of inferences reported below are the mean values for the 6 runs of a particular combination. The reported mean inference ratios and example ratios (calculated in the same manner as inference ratios) are the mean of the inference ratios and example ratios calculated for each run. Therefore, they do not correspond exactly to the raw inference and example means.[4]

7.1. The effect of irrelevant example features

Method. In our first experiment, we examined eight degrees of irrelevant features in the example descriptions by setting the padding factor parameter to have values from 1.0 to 2.4, in steps of 0.2. A 1.0 value means that no irrelevant features are included in the example descriptions, while a 2.4 value means that the ratio of irrelevant to relevant features was 1.4. The eight levels of irrelevant features were crossed with target-concept levels 2, 3, 4, and 5.

Results and Observations. For brevity's sake, we have presented data for only four ratios of irrelevant to relevant features: 0, 0.4, 1.0, and 1.4 in Table 6. (The interested reader can contact us for complete data sets for all experiments reported here.) The inference ratios indicate that LAIR's deductive advantage over the closure system increased as the percentage of irrelevant features increased, particularly for level-5 target concepts. The data on raw number of inferences indicate that increasing the percentage of the irrelevant features in level-5 concept examples had a much larger impact on the closure system than it did

Table 6. Inferences and examples to criterion for different degrees of irrelevant features in example descriptions.

		Inferences			Examples		
		LAIR	Closure	Ratio	LAIR	Closure	Ratio
Concept Level	Irrel./Rel. Features[a]						
2	0.0	6.3	13.6	.32	10.0	10.0	.50
	0.4	16.3	24.5	.40	13.0	10.3	.53
	1.0	50.7	132.2	.32	32.7	32.7	.50
	1.4	50.2	165.7	.30	40.0	36.0	.52
5	0.0	515.3	1824.2	.25	51.7	51.7	.50
	0.4	659.2	4277.2	.18	58.3	58.3	.50
	1.0	967.0	4641.2	.20	87.0	54.0	.57
	1.4	1434.2	11963.7	.12	139.0	115.0	.54

[a]Ratio of irrelevant to relevant features in each example. The value 0 means no irrelevant features.

on LAIR. For level-5 concepts, the closure system made 6.5 times as many inferences (1824.2 vs. 11963.7) as the irrelevant feature ratio increased from 0 to 1.4, while LAIR had a 2.7 factor increase (515.3 vs. 1434.2).

In terms of trials to criterion, we had expected that LAIR would be at a large disadvantage relative to the closure system and that the disadvantage would increase as the percentage of irrelevant features increased. This is because the closure system simply gets more mileage out of each example and therefore should be able to identify irrelevant features faster than LAIR can. In addition, a 1.4 irrelevant-feature ratio nearly ensures a large overlap of irrelevant features in the example set. This would increase the likelihood that an irrelevant feature in the remembered past positive example would reoccur in subsequent positive examples, leading LAIR to retain irrelevant features in its concept hypothesis for a longer time, thereby increasing its trials to criterion.

The results did not support these expectations in any consistent way. For about half the entire set of runs, the example ratios were 0.5, indicating that the two systems required the same number of trials to criterion. For the remaining runs, LAIR required 10–25% more examples than the closure system. One conjecture as to why LAIR did not perform as badly as expected is that a high degree of overlapping irrelevant features would occur in negative examples as well as positive examples, making it easy for the system to dispose of them. This conjecture suggests that most of LAIR's difficulty with irrelevant features would occur on intermediate padding factors, for which there might be less overlap among positive and negative examples. In fact, this is when LAIR did need more examples to reach criterion than the closure system.

The Table 6 data are representative of one general trend that holds true of all the experiments we ran. The higher the target-concept level, the more inferences and examples are needed to learn the target concept, although the trend across levels is not always perfectly linear. This is because a particular level-3 concept could, by chance, have a more difficult definition to learn (e.g., it might have more negated antecedents or have closer overlap with other level-3 concepts) than a level-4 concept. Generally speaking, however, inductive and deductive effort increases as a function of target-concept level for both LAIR and the closure system.

7.2. The role of information conservation

Method. For our second experiment, we investigated the relative benefits of four different information conservation schemes: no information conservation, conserving information for required features dropped from the concept hypothesis (*conserve-required*), conserving information for both the dropped required features and the unique example features (*conserve-req+curr*), and maximum conservation (*conserve-max*).

These methods differ only in how they constrain the search for generalized features for information conservation. In the no-conservation case, deduction is used only to reconcile the current concept hypothesis with the current example. No conservation of dropped features is done. The conserve-required and the conserve-req+curr (the standard version of LAIR) cases operate according to the two constraints described in the earlier section on information conservation: The search for generalized features of some required stack feature (a) continues only when it make heuristic sense to do so (see note 2), and (b) stops as soon as one generalization can be added to the concept hypothesis. The search under the conserve-max method is controlled by the first constraint, but it terminates when the required stack is empty rather than when the first generalization is found.

Pilot experiments indicated that information conservation methods may interact with levels of irrelevant features, because the latter factor has such a pronounced affect on the deductive effort. We crossed the extreme levels of irrelevant features (padding factors 1.0 and 2.4) with the extreme target concept levels (levels 2 and 5), creating a 4 (conservation method) $\times$ 2 (target concept level) $\times$ 2 (degree of irrelevant factors) design.

Results and Discussion. The results of this experiment are presented in Table 7. First consider the data when there are no irrelevant features (ratio of irrelevant to relevant features is 0.0). In this case, the type of conservation scheme that LAIR uses does not seem to matter much: relative to complete closure, any controlled information conservation method reduces inferences at no or little cost in examples to reach criterion. However, when no conservation is done, LAIR makes fewer inferences than the closure system (e.g., for level-2 concepts, inference ratio = 0.37) but requires considerably more examples to reach criterion (example ratio = 0.63). Thus, conservation does play an important role within the LAIR framework to speed learning.

The trends are more pronounced when the ratio of irrelevant to relevant features increases to 1.4. Consider LAIR's learning data when it learned level-5 target concepts with no conservation versus all other conservation schemes. When no conservation is done, LAIR made about 25% fewer inferences to learn level-5 target concepts (1317.0 inferences vs. a mean of 1620.0 inferences for the three conservation methods), but required about 17% extra examples (129.3 vs. a mean of 109.6 for the three conservation methods). This also confirms that information conservation, while costing inferences, does promote faster learning when measured by trials to criterion.

The inference ratios show that LAIR made fewer inferences than the closure system on level-5 target concepts. Under the most difficult learning case—level-5 concepts with a 1.4 ratio of irrelevant to relevant features—the closure system made 3—6 times more inferences (over 9000 mean inferences to LAIR's average of about 1600), but used about 25% fewer examples, depending on the conservation scheme LAIR used. Again, it takes LAIR longer to recognize all the irrelevant features in these high-level concepts, because

Table 7. Inferences and examples means and ratios for different conservation methods and amount of irrelevant features.

	Inferences			Examples		
	LAIR	Closure	Ratio	LAIR	Closure	Ratio
Method						
Irrel./Rel. Features=0						
Concept Level = 2						
None	18.7	24.0	.37	23.7	13.0	.63
Req	20.3	24.0	.40	17.3	13.0	.52
Req+Curr	21.3	24.0	.41	17.3	13.0	.53
Max	20.3	24.0	.40	17.3	13.0	.52
Concept Level = 5						
None	60.7	397.7	.13	28.7	22.7	.54
Req	131.8	398.7	.29	22.7	22.7	.50
Req+Curr	138.8	397.7	.30	22.7	22.7	.50
Max	132.8	397.7	.30	22.7	22.7	.50
Irrel./Rel. Features = 1.4						
Concept Level = 2						
None	27.0	37.6	.32	44.3	20.7	.61
Req	33.3	37.8	.43	38.7	20.7	.58
Req+Curr	33.5	37.8	.43	38.7	20.7	.58
Max	33.5	37.8	.43	38.7	20.7	.58
Concept Level = 5						
None	1317.0	9525.0	.12	129.3	88.0	.57
Req	1638.0	9525.0	.14	112.3	88.0	.55
Req+Curr	1612.5	9525.0	.14	109.7	88.0	.55
Max	1611.2	9525.0	.14	106.7	88.0	.53

it does less deductive work on each example description. When the number of irrelevant features is low, LAIR compares quite favorably with the closure system on both trials to criterion and number of inferences.

These results suggest that the most appropriate constructive induction method for a particular task can be chosen by considering the level of irrelevant features in the example descriptions, distance of the concept definition from these descriptions, and whether it is important to optimize inferential effort or examples used. The data also indicate that information conservation, which can be viewed as a heuristically-guided form of constructive induction, speeds learning.

7.3. The effect of positive and negative training bias

Method. In this experiment, we examined three different types of training sequences: a *random sequence*, in which there was a 50% probability that any training example would be positive or negative, a *positive-bias* sequence, in which there was a 90% probability that any training example would be positive, and a *negative-bias* sequence (the converse of the positive-bias sequence). The remaining parameters were set at the default values given in Table 5.

Results and Discussion. The inference and example ratios given in Table 8 indicate that

LAIR consistently made fewer inferences than the closure system, and in most cases reached criterion with fewer or about the same examples. There is one case (learning level-4 target concepts with a negative-bias sequence) for which LAIR used considerably more examples, in our consideration, than the closure system (example ratio = 0.58). We regard this as an unsystematic fluctuation, because the example ratio for level-5 target concepts drops to 0.50 for this condition. Hence, the increase for level-4 target concepts is inconsistent with the general trend that difficulty increases as the level of target-concept increases. Such outlying data points are typically due to one or two particularly difficult concepts (i.e., with more negative features or high overlap with other definitions) within the six independent runs.

Not surprisingly, both systems required more examples and more inferences to reach criterion with a negatively-biased sample. However, an interesting result is that LAIR made fewer inferences with a negatively-biased sample than with a positively-biased sample (e.g., 308.7 versus 528.0 for level-5 concepts), but the reverse was true for the closure system (e.g., 2823.0 versus 722.2). Usually, the number of inferences is positively correlated with the number of examples needed to reach criterion, but LAIR used more examples with a negatively-biased sequence than it did with a positively-biased sequence.

Why would LAIR make fewer inferences but need more examples with a negatively-biased sample? One potential explanation is the following. When the training sequence consists primarily of negative examples, most trials will be specialization attempts. With very few positive examples to suggest potential required features, it is likely that a specialization attempt will succeed by adding a forbidden feature to the concept hypothesis. One constraint on the add-forbidden operator is that a new forbidden feature cannot be true of some previous positive example. Seeing relatively few positive examples, LAIR will have marked only a few features in the knowledge base with the *can't-forbid* tag, so any of a number of these specialization steps looks good. If a specialization must be derived, inferencing expended in the search for and proof of specializations can also probably stop sooner for the same reason: very few derived features will have appeared in any positive instance, so they too would seem to be valid forbidden features. The problem is that incorrect specializations will go undetected until a positive example is encountered, which infrequently happens. Thus, LAIR can make (incorrect) specializations with fewer inferences on each trial, but will require more examples to see positive examples that can correct these bad revision steps.

Now consider the alternative case. In a positively-biased sample, LAIR is primarily applying the add-required operator, which triggers the proof procedure for testing whether the current concept description covers the positive example and for doing information conservation. As the algorithm in Table 2 indicates, the cycle is one of applying the add-require operator and (possibly) applying the drop-require operator. When a negative example finally occurs, LAIR will expend more inferential effort to find specialization features in the domain theory that have not been marked with *can't-require* or *can't-forbid* tags from all the previous experience with positive examples. However, once a discriminating feature is found, it is more likely to be a correct revision to the concept description. Hence, LAIR will make more inferences per positive example, but require less examples to reach criterion.

Table 8. Inference example means and ratios for training sequences with different types of bias.

	Random			Positive Bais			Negative Bias		
	LAIR	Closure	Ratio	LAIR	Closure	Ratio	LAIR	Closure	Ratio
Concept Level									
Inferences									
2	18.0	51.5	.35	49.8	31.3	.46	38.0	138.2	.33
3	191.2	105.2	.49	232.0	157.0	.49	86.3	286.7	.23
4	124.8	486.8	.25	125.5	456.5	.26	152.5	793.8	.22
5	360.8	1341.8	.38	528.0	722.2	.45	308.7	2823.0	.23
Examples									
2	13.0	16.0	.44	31.3	12.7	.55	60.3	57.7	.52
3	46.0	23.7	.55	52.0	43.3	.52	58.0	69.3	.42
4	21.3	39.0	.37	13.7	42.7	.34	82.0	63.3	.58
5	57.7	31.7	.54	34.3	20.0	.54	85.0	80.7	.50

The opposite trends occur for the closure system. Given a positively-biased sample, it also engages in mostly add-require and drop-require operations. For each feature that is dropped from the concept description, the closure system marks all features derivable from that feature and the domain theory as ones that cannot serve as required features on future revision steps. With a negatively-based sample, the closure system, like LAIR, will stop at the first specialization that discriminates the negative example. Also, like LAIR, the closure system will need fewer examples given a positively-biased sample than given a negatively-biased sample. However, the closure system does not save any inferences by stopping at the first specialization, since it always computes all possible inferences for an example. The closure system makes more inferences given a negatively-biased sample because it needs more examples to reach criterion when confronted with that kind of example bias.

The random sample seems to provide the best tradeoff for a system like LAIR in terms of average number of inferences and average number of examples. However, the important general point of these findings is an appreciation of a constructive induction method's sensitivity to training sequence biases. For LAIR's method, a positively-biased sample forces more inferences per example, which ultimately reduces the number of examples necessary to reach criterion.

7.4. The effect of knowledge-base width

Method. To explore the effect of knowledge-base size, we constructed eight knowledge bases that had different widths, ranging from 30 concepts per level to 100 concepts per level, in steps of 10. The width was constant across all five levels within a knowledge base. We crossed these eight widths with target-concept levels 2 and 5, for an 8 × 2 design.

Results and Discussion. For brevity's sake, we present data for widths 30, 60, and 100 in Table 9. These results are typical of the general trend across the eight levels. Regardless of width, LAIR consistently made fewer inferences than the closure system and typically required about the same number of examples to reach criterion on both level-2 and level-5 target concepts.

Table 9. Inferences and examples means and ratios for different knowledge-base widths.

	Inferences			Examples		
	LAIR	Closure	Ratio	LAIR	Closure	Ratio
Concept Level = 2						
Width						
30	109.5	351.5	.35	50.3	20.7	.57
60	95.8	89.0	.40	32.0	15.0	.56
100	13.7	17.0	.44	5.3	5.3	.50
Concept Level = 6						
Width						
30	488.8	1780.7	.29	102.0	49.0	.61
60	128.3	381.5	.32	22.7	22.7	.50
100	99.3	257.5	.35	18.0	18.0	.50

The interesting results, however, concern the effect of width on absolute inductive and deductive effort. Generally speaking, the narrower the knowledge base, the more inferences and examples each system required (although there were some deviations due to variations in concept difficulty). When the knowledge base is narrow (e.g., 30 concepts per level), the concept definitions have a larger degree of overlap at each level as a result of how antecedents are selected from lower levels in the knowledge base to form a higher-level rule definition. This increases the likelihood that any particular feature participates in many overlapping rule definitions. Thus, a system doing complete deductive closure is likely to derive many more features under this high concept-definition overlap condition. The other effect of a narrow knowledge-base width is that the positive and negative examples for any concept will also tend to have many features in common. This will increase the search and deductive effort to find features that can be added as legal specializations when a negative example is encountered.

Conversely, when the knowledge base is wide, positive and negative examples are likely to contain less overlap. This makes it easier to find correct discriminating features for specialization steps. In addition, the apparent dissimilarity between two very different positive instances can only be resolved during long chains of inferencing, i.e., they force very large steps through the search space during proof procedures. Thus, the correct concept description can be identified faster, with less deductive effort, under these conditions. The following experiment on knowledge-base shape offers some further clarification of these search space characteristics on constructive induction.

7.5. The effect of knowledge-base shape

Method. For our final experiment, we created different knowledge base shapes by varying the number of features at each level. On the basis of pilot experiment results, we focused on three general shapes: a square-shaped knowledge base, which had 65 concepts at each level; a pyramid-shaped knowledge base, which had 30-40-50-60-70-80 concepts at levels 0-5, respectively, and an inverted pyramid shape, which had 80-70-60-50-40-30 concepts at levels 0-5, respectively.

Table 10. Inference example means and inferences for different knowledge-base shapes.

	Shape								
	Square			Pyramid			Inverted Pyramid		
	LAIR	Closure	Ratio	LAIR	Closure	Ratio	LAIR	Closure	Ratio
Concept Level									
Inferences									
2	13.0	18.8	.41	37.8	29.8	.50	34.8	76.7	.42
3	50.2	90.2	.38	25.0	37.8	41	151.2	235.3	.34
4	59.3	168.5	.33	49.3	93.7	.39	327.7	1499.2	.22
5	286.7	1240.8	.23	246.7	736.2	.29	936.2	6517.8	.16
Examples									
2	13.3	13.3	.50	34.3	24.7	.55	24.7	19.7	.55
3	21.3	21.3	.50	11.0	11.0	.50	53.3	26.0	.56
4	12.7	12.7	.50	11.3	11.3	.50	67.7	40.2	.56
5	42.0	42.0	.50	28.7	28.7	.50	70.3	70.0	.50

Results and Observations. The data for this experiment are presented in Table 10. LAIR consistently made fewer inferences than the closure system for all shapes and this advantage increased as a function of target concept level. Knowledge-base shape did not affect the degree of this advantage on inferences.

For the square and pyramid shapes, both systems required about the same number of examples to reach criterion. The trends were not as consistent for the inverted pyramid: LAIR needed about as many examples as closure for target concept level 5, but needed 50% more examples for the lower levels. There is no obvious explanation for these data, other than perhaps the idiosyncratic affect of particularly difficult target concepts at those levels.

LAIR showed a distinct advantage over the closure system when learning level-5 concepts in both the pyramid-shaped knowledge base and the inverted pyramid-shaped knowledge base. These two scenarios present qualitatively different search spaces and the reasons for LAIR's advantage in each case are different. First consider the inverted-pyramid shape. Of the three shapes, the inverted pyramid was uniformly the most difficult knowledge-base shape for both LAIR and the closure system. The fan-out of this shape as level increases promotes an explosion of inferences associated with level-5 concepts. The closure system suffered more than LAIR did from this inference explosion, making about 55 times more inferences to learn level-5 target concepts than it did to learn level-2 target concepts (6517.8 vs. 76.7). LAIR was similarly affected, but "only" by a factor of 27 (936.2 vs. 34.8), precisely because of the heuristic control in the inferencing process.

The second situation where LAIR showed a large advantage over closure was learning level-5 concepts from the pyramid-shaped knowledge bases. LAIR made about a third of the inferences while reaching criterion in the same number of examples. The narrowing shape of the pyramid produces a higher-degree of overlap in the definitions of level-5 concepts relative to other levels. Recall that this happens because, as the concept population decreases with increasing level, our method of selecting antecedents for level-5 rule definitions has a more constrained set of possibilities at level-4. However, the examples are constructed by backward chaining through levels that become wider and wider (and hence

have less concept definition overlap), increasing the likelihood that there will not be much overlap of the level-0 primitives appearing in the resulting examples. This result supports the conjecture that LAIR learns fastest when there is little overlap among positive examples and among negative examples.

The conjecture that LAIR learns particularly fast with non-overlapping examples is further confirmed by comparing the examples-to-criterion needed to learn level 3–5 concepts from a pyramid knowledge base with examples-to-criterion for the square or inverted-pyramid shapes. For example, LAIR averaged 11 examples to learn level-3 target concepts from a pyramid knowledge-base, but needed 21 and 53 examples given square or inverted pyramid shapes, respectively. Again, the wide level-0 base of the pyramid shape, relative to these other two shapes, increases the likelihood that example descriptions will have fewer overlapping features.

Taken with the results from the knowledge-base width experiment, these data indicate how abstract characteristics of a domain theory—namely its characterization as a search space in terms of the density of features and definition overlap—can impact both the deductive and inductive effort made by a constructive induction system.

8. Theoretical evaluation of LAIR

Recent theoretical work in machine learning (Valiant, 1984) provides another method of evaluating LAIR. In this section, we consider the theoretical criteria by which an algorithm fits the pac-learning model and the domain theories for which our system meets these criteria.

Blumer, Ehrenfeucht, Haussler, and Warmuth (1986) define an algorithm to learn from examples if it can, in a feasible (polynomial) amount of time, find with confidence $1\text{-}\delta$ a rule that is reasonable accurate. Reasonably accurate means that the rule found by the algorithm will be able to predict future events drawn from the same distribution on which it has learned, with accuracy $1\text{-}\epsilon$. No assumption is made about the input distribution of examples, except that it is time-invariant. This formulation by Blumer et al. (1986) is a fairly standard version of Valiant's (1984) original definition of pac-learning.

LAIR can be modified to meet these criteria of a pac-learner when working with domain theories and instance spaces that have the following properties: (a) there are a finite number of features, n, in the instance and derived feature space, (b) the amount of time spent in deductively matching any feature never takes more than some fixed polynomial time, and (c) there is a target concept description with a conjunctive representation using only the features in the instances and the derived features.

It is possible to convert LAIR into a polynomial pac-learner because it modifies its current concept description, i.e., current hypothesis, only a polynomial number of times. Recall that because of the can't-require and can't-forbid constraints on the inductive operators, LAIR can add (and possibly drop) a feature as a required or a forbidden at most once. If n is the number of features, LAIR can make at most $2n$ adds or drops.

Let $x = 2n$ and h_i be the ith hypothesis that LAIR forms during learning. Then LAIR forms $t \leq x$ (not necessarily distinct) hypotheses $h_1, h_2, \ldots h_t$. The number of times that the current hypothesis predicts the next instance incorrectly is at most x and whenever this occurs, a new hypothesis is formed.

Following the approach taken by Angluin (1987) and Kearns et al. (1987), we can convert LAIR to a pac-learning algorithm, LAIR′, by testing hypotheses as they are formed. A test consists of presenting LAIR′ with a new example. The current hypothesis passes the test if LAIR′ does not modify the hypothesis after seeing the example. The general form of algorithm LAIR′ is as follows:

1. Initialize LAIR with the first example and form the initial hypothesis h_1.
2. i ← 1.
3. Run LAIR on at most $\frac{1}{\epsilon} ln \frac{x}{\delta}$ randomly-generated examples until either
   case 3a: LAIR changes its hypothesis to h_{i+1}, in which case do
      3a.1. i ← i + 1.
      3a.2. Go to step 3.
   case 3b: LAIR keeps h_1 for all $\frac{1}{\epsilon} ln \frac{x}{\delta}$ tests, in which case output h_i and halt.

PROOF: LAIR′ pac-learns after $m \leq \frac{1}{\epsilon} ln \frac{x}{\delta}$ examples. If LAIR′ outputs h, it must have been consistent with at least $\frac{1}{\epsilon} ln \frac{x}{\delta}$ randomly-generated examples, because otherwise h would have been changed. A hypothesis is defined to be ϵ-BAD if the probability of its error is $>=\epsilon$. Let p_i denote the probability that a given h_i is both ϵ-BAD and consistent with $\frac{1}{\epsilon} ln \frac{x}{\delta}$ examples (and hence output in step 3b). Then

$$p_i \leq (1-\epsilon)^{\frac{1}{\epsilon} ln \frac{x}{\delta}} \leq \frac{\delta}{x}.$$

So the probability p that LAIR′ outputs *any* ϵ-BAD hypothesis is

$$p \leq \sum_{i=1}^{x} \text{probability } p_i \text{ LAIR}' \text{ outputs } h_i \wedge h_i \text{ is } \epsilon\text{-BAD}$$

$$\leq \sum_{i=1}^{x} \text{probability } h_i \text{ passes } \frac{1}{\epsilon} ln \frac{x}{\delta} \text{ tests} \wedge h_i \text{ is } \epsilon\text{-BAD}$$

$$\leq x \left(\frac{\delta}{x}\right)$$

$$\leq \delta.$$

Thus with probability $\geq$ 1-δ, the hypothesis output by LAIR′ has accuracy $\geq$ 1-ϵ. ■

9. General discussion

We have presented an approach to controlling constructive induction by reducing it to a deductive step done in service of the specific concept revision goals of incremental learning from examples. These goals are to generalize or specialize the concept description as necessary when a new example is encountered. To achieve these goals, LAIR searches

the domain theory for features outside the representation space defined by the example features. These features are then introduced into the concept description, thus changing the representation space for the concept hypothesis.

In addition, we proposed an information conservation scheme through which LAIR attempts to preserve information about features that it must remove from the concept description, as well as features that are idiosyncratic to some positive example. It does this by searching for generalizations of those features that would constitute valid additions to the concept description, as determined by the preconditions of the inductive operators.

For this kind of learning—incremental modifications to a single concept description with no reprocessing of past examples—we have shown that constructive induction can be guided by the requirements of the learning task and controlled by constraints on the inductive operators. Deduction is done on an as-needed basis. It is initiated by the preconditions of inductive operators and hence is done to meet the very constrained goals of reconciling the concept description with the current example description (and, in our framework, of conserving information about idiosyncratic features in the current example or current concept).

Given these requirements, there is no need to infer all possible extensions to an example description. In fact, it may be counterproductive to do so and incorporate them into the concept description. The learner cannot know, on the basis of a single example, whether it is relevant or valid to derive all possible extensions of the example description and incorporate them into the concept description. The necessary degree of deduction, generalization, or specialization can be known only as each additional example is encountered. Furthermore, the addition of all consistent higher-level features to the concept description may need to be undone at some later point, again with much deductive effort, when subsequent examples are seen.

The results on absolute and relative number of inferences for LAIR and the closure system support these claims. The closure system was often much more sensitive to the impact of our learning scenario manipulations than was LAIR. This was evident in the irrelevant features experiment, for example. The most demanding case (maximum degree of irrelevant features for level-5 target concepts) was seven times more difficult for the closure system than the easiest case (no irrelevant features for level-5 target concepts) and 2.7 times more difficult for LAIR.

The other important finding from our empirical investigations concerned examples to criterion. In general, LAIR compared quite favorably to deductive closure on this measure. Because the closure system makes all possible deductions for each example, it gets more mileage out of each example. This means that the deductive closure version has always ruled out at least as many hypotheses as LAIR, and, in theory, should learn faster. However, while LAIR never learned faster than the closure system, a conservative summary of all runs across all experiments is that it learned as fast at least 50% of the time; for the remaining cases, it usually needed about 25% more examples. We think that these findings are good evidence for the usefulness of some kind of information conservation that is heuristically-guided and well short of deductive closure.

In addition to specific evaluations of LAIR relative to deductive closure, we identified several learning scenario features that affect constructive induction. The most interesting results involved the sensitivity to training sequence bias and the knowledge-base shape and

width. Knowing the effects of the training sequence bias has general implications for supervised learning when items are specifically chosen by a teacher. The results on knowledge-base shape and width are also important, although the particular trends are specific to the systematic ways in which concept definitions were automatically generated for our knowledge bases.

In the remainder of this section, we discuss LAIR's relation to other construction induction methods and learning paradigms, as well as ways in which this framework can be extended to be more efficient and general in its application of domain knowledge.

9.1. Learning from examples using constructive induction

Michalski's (1973) AQ and Larson and Michalski's (1977) INDUCE were two of the first algorithms to use constructive induction to a logic-based language, with an explicit domain theory for introducing new features into the language. Michalski (1983) also formalized constructive generalization as an inductive rule, and posed the problem of controlling constructive induction. Both AQ and INDUCE carry out constructive induction by first applying the domain rules to the examples to compute their deductive closure prior to induction, and then providing these specialized example descriptions to the induction engine.

Like AQ, Sammut and Banerji's (1986) MARVIN requires computing the deductive closure prior to learning, which distinguishes it from our incremental, deduction-as-needed approach. However, it incrementally introduces these new features into the language by a succession of replacement operations, using queries of the teacher to determine if these operations are correct.

Although both MARVIN and LAIR use prior knowledge to learn conjunctive concepts, the research goals of the two systems are somewhat different. The critical distinction between MARVIN and LAIR is the use of the domain theory during the learning process. MARVIN computes all possible replacements each time it attempts to form a new hypothesis. This is essentially equivalent to considering the entire domain theory. As our experiments demonstrated, this can be expensive if there are many pieces of knowledge that are implied by the current description and the knowledge base. With incremental learning, we let the requirements of the learning task dictate when and how much deduction should occur to reconcile two descriptions.

Drastal, Raatz, and Czako (1989) describe a system called MIRO, which uses a constructive induction technique based on the construction of an abstract language for deriving a concept description. The abstract language is a set of features that can be derived from a domain theory and the instance language used to describe training examples. They begin inducing the concept description by using only the most general features from the abstract language. Additional, more specific features from the abstract language are added to the concept description as required, until a concept description is produced that covers all positive cases and excludes all negative cases. Drastal et al.'s empirical results show that using an abstract language of derivable domain theory features, rather than the instance language, to produce the concept description leads to faster learning when there is little apparent syntactic overlap among training instances. Without the use of a domain theory, their algorithm must identify a complex disjunctive form of instance features that,

for some training sets, can become computationally infeasible. MIRO is more powerful than LAIR because of its ability to learn disjunctive, as well as conjunctive, descriptions over features in the transformed space, and its ability to deal with noisy examples (Drastal, Meunier & Raatz, 1989).

This work differs from LAIR in its concern with the use of constructive induction for non-incremental learning and its search from most abstract to most specific. MIRO's abstract-to-specific search requires that the deductive closure of the examples be computed, which is not necessary in LAIR. The constructive induction aspect of MIRO's method is the derivation of the abstract language, which is done prior to the actual induction process. The feature space is transformed (actually, redefined) before rather than during learning. Thus, this approach presupposes that a completely reformulated feature space is required to converge on a reasonable concept description with a reasonable amount of computational effort. This requirement is necessary, as their experimental results show, when there is little overlap among instances in the training set. Our approach in LAIR is quite different, in part because of our focus on incremental learning. LAIR uses constructive induction, in the sense of introducing domain theory features into the representation space, only when it needs to. The deductive component is driven solely by the difficulty of the learning task and is not a fixed part of the inductive algorithm. LAIR could easily learn concept descriptions that did not necessitate domain knowledge without a change to the framework—there would simply be no trigger of long deductive proofs if the concept revision processes could proceed on syntactic adjustments.

9.2. Explanation-based generalization and constructive induction

Explanation-based learning is a method that, given a goal concept and a positive example of the goal concept, uses a domain theory to construct a proof that "explains" that the example satisfies the goal concept description (Mitchell, Keller & Kedar-Cabelli, 1986; DeJong & Mooney, 1986). By virtue of constructing an explanation, this method identifies what features of the example's description are relevant to being an exemplar of the concept. A generalization procedure then finds the weakest conditions for which this explanation holds. The default explanation-based learning approach assumes a complete and correct domain theory, but recent research on this paradigm has begun to address the problems of learning when these conditions do not hold. Some of the systems that address these problems are related to LAIR.

From the explanation-based learning perspective, LAIR can be viewed as a system that corrects a target concept that is non-existent or incorrect. In particular, LAIR is related to the methods for multiple-example explanation-based generalization (mEBG) developed independently by Kedar-Cabelli (1985), Hirsh (1988), Pazzani (1988), and Cohen (1988). Flann and Dietterich (1989) describe these systems, and their own refinement, which they call *induction over explanations*. The essential idea behind these systems is to prove a target concept description from a set of examples, and then to use the leaf nodes of the shared part of the proof tree as an alternative concept definition. Like LAIR, existing mEBG systems can only learn concept descriptions that have a conjunctive representation in the instance and derived feature space, as otherwise there may not be any shared structure. Unlike

existing mEBG systems, LAIR can construct a concept description without being given a target concept description, and it can both generalize or specialize a target concept description that is incorrect. Flann and Dietterich's (1989) induction over explanations is capable of specializing an overly-general target concept description, but does not appear capable of generalizing an overly-specific concept description. However, their method learns in a first-order language, so it is solving a more difficult problem than LAIR.

Like mEBG systems, LAIR will tend to find concept descriptions that consist of the least abstract ("most operational") features. This choice is explicitly enforced in mEBG systems by choosing the leaf nodes of the common subtrees as the new concept description. In LAIR, it is a side-effect of the specific-to-abstract controlled constructive induction strategy. However, there is no notion of an operationality criterion in the LAIR paradigm, hence the system can derive concept definitions that use features not appearing in the instance descriptions.

Justifications for multiple examples may give a system like LAIR the ability to formulate disjunctive definitions using only primitive predicates. Consider the three examples of cup: balanced-cup, empty-cup, and tea-cup. Each of them meets the concept definition's requirements of liftable, open-vessel, and stable in slightly different ways. This is revealed in their proof structures; e.g., two are liftable because they are graspable, but one is graspable because it has a handle, the other because of a conjunction of features like cylindrical, small, light, and not-hot. It would be possible to combine these derivations to produce a description of cup that unpacks each high-level predicate in such a sway that captures these different reasons, e.g., [[handle or [cylindrical & small & light & not-hot]] & . . .]. While LAIR cannot directly learn a disjunctive concept, it can remember examples it has been given for a concept, and then reformulate their proofs into a disjunct of primitive predicates. This would give LAIR something analogous to an operational definition for a concept as described by Mitchell et al. (1986).

9.3. Directions for future work

In a system like LAIR, the domain theory is either given to the system, or acquired during prior learning. Alternatively, a system can attempt to acquire a domain theory, by constructing new features, while trying to learn a concept. This kind of learning is closely related to conceptual clustering, and Holder (1989) has implemented a conceptual clustering system with a constructive inductive component.

In most feature-construction systems, constructive induction is rarely controlled directly. Instead, the system controls the generation of features and typically all generated features are used exhaustively (e.g., Rendell, 1985; Pagallo and Haussler, 1988; Matheus, 1989). Because control of feature generation is the crucial problem, the techniques used are quite different than the ones used in LAIR. For example, Muggleton's (1987) DUCE system uses a set of transformation operators to introduce new features and relies on an external teacher to indicate whether the proposed features are meaningful. Feature construction in Rendell's PLS0 (1985) is guided by a clustering algorithm that identifies new features based on a domain-defined utility measure. This method is applied in successive stages, in which each stage constructs new features from those verified at the preceding stage. One direction of

future work includes exploring whether feature-construction methods can be incorporated into the LAIR framework, i.e., using an incremental approach guided by the immediate goals of the learning-from-examples task.

When a domain theory and deduction play a role in any task, one becomes faced with controlling the amount of inferencing done by the system, or at least directing it in a way that might minimize it. A system like LAIR acquires and extends its domain theory by a sequence of induction tasks. However, each time LAIR adds a just-learned concept description to its knowledge base, it forgets the kinds of inferences and deductions it made to acquire that concept. In addition to learning the contents of the domain theory (i.e., the new concept definitions), the system might also "learn about learning" in the domain. This would include attending to what types of knowledge have been frequently used to learn related concepts. Presumably this could focus and hence reduce the time it takes to make the relevant deductions.

There are a number of possible mechanisms one could explore that might serve this purpose. Deductive learning methods, such as explanation-based learning, knowledge compilation (Anderson, 1986), and chunking (Laird, Rosenbloom & Newell, 1986) would be appropriate for restructuring frequently-accessed sequences of inference rules into larger units of knowledge. Using this type of mechanism would introduce a kind of bias shift (Utgoff, 1986), in that the inferences and hypotheses initially explored would be determined by the compiled knowledge structures. Alternatively, one could simply treat learning as a type of problem-solving activity that generates positive and negative examples (effective learning vs. ineffective learning). These concepts descriptions, and their corresponding rules, could be used as meta-rules to control LAIR's inferencing.

9.4. Summary

We believe that the framework we have explored in the LAIR system demonstrates how deductive and inductive techniques can be used to apply domain knowledge to learn new concept descriptions. The benefit of our approach is that the requirements of the inductive task constrain when and how constructive induction should be done. Relative to our instantiation of deductive closure, the LAIR system and its information conservation process was shown to greatly reduce the number of inferences needed to learn a concept to criterion, at little or no cost to the number of necessary examples. We think this approach is useful for learning situations in which a domain theory is necessary to make sense of seemingly disparate, but otherwise complete examples. The empirical evaluation also demonstrated the importance of assessing a constructive induction method's sensitivity to variable characteristics of the domain theory viewed as a search space, definable by features such as concept definition overlap, density of features, and fan-in and fan-out inference chains.

Acknowledgments

This work was supported by Natural Sciences and Engineering Research Council of Canada operating grant A0089 to Renée Elio. The order of the authors is arbitrary. The core ideas

of this research were presented at IJCAI-87 and were part of work submitted by Larry Watanabe in partial fulfillment for a Masters of Science Degree in Computing Science at the University of Alberta. We would like to thank an anonymous reviewer and especially Pat Langley for extensive suggestions that greatly improved an earlier version of this manuscript and Leonard Pitt for his advice on the theoretical evaluation. Jeff Pelletier, Randy Goebel, and Ryszard Michalski also provided useful comments on this work. Requests for reprints can be sent to Larry Watanabe, Beckman Institute, University of Illinois, 405 North Matthews, Urbana, IL 61801, watanabe@cs.uiuc.edu.

Notes

1. Gennari, Langley, and Fisher (1989) have referred to this as an *incremental hill-climbing approach* to learning.
2. The perturbation parameter was set at 0.1 for all experiments reported here. Functionally, that meant that by changing approximately 10% of the negative example's descriptions, it could be converted to a positive instance. Thus, negative examples were not near misses, by Winston et al.'s (1983) definition, but neither were they very "far" misses.
3. Recall that, for information conservation, LAIR uses some feature P from the required stack to search for new required candidates only if P is true of either the current positive example or the remembered past positive example. For specialization steps, the system uses some feature P from the required stack or forbidden stack to search for new required and forbidden candidates only if P is a discriminating feature: provable of one example and not provable of the other.
4. Care must be taken in interpreting the absolute values of the raw inference and example means, because the standard deviations for these measures were quite high. This is because there is considerable variation in the difficulty of the target concepts, even within one level. However, the mean inference and example ratios have very low standard deviations, because a ratio is computed for each separate target concept that the two systems learned. They are a stable measure of the relative performance of LAIR vs. the closure system on a particular target concept, but clearly do not reflect the absolute effort made by each system. This is why we include the raw means as well.

References

Anderson, J.R. (1986). Knowledge compilation: The general learning mechanism. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Angluin, D. (1987). Queries and concept learning. *Machine Learning, 2*, 319–342.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. *Proceedings of the 18th Annual ACM Symposium on Theory of Computation* (pp. 273–282). Berkeley, CA: Association for Computing Machinery.

Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence, 36*, 149–176.

Cohen, W.W. (1988). Generalizing number and learning from multiple examples in explanation based learning. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 256–269). Ann Arbor, MI: Morgan Kaufmann.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*, 147–176.

Drastal, G., Meunier, R., & Raatz, S. (1989). Error correction in constructive induction. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 81–83). Ithaca, NY: Morgan Kaufmann.

Drastal, G., Raatz, S., & Czako, G. (1989). Induction in an abstraction space: A form of constructive induction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 708–712). Detroit, MI: Morgan Kaufmann.

Flann, N.S. & Dietterich, T.G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4,* 187–226.

Gennari, J.H., Langley, P. & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence, 40,* 11–61.

Hirsh, H. (1988). Knowledge as bias. *Proceedings of the First International Workshop on Change of Representation and Inductive Bias* (pp. 186–192). Briarcliff, NY: AAAI.

Holder, L.B. (1989). Empirical substructure discovery. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 133–136). Ithaca, NY: Morgan Kaufmann.

Kearns, M., Li, M., Pitt, L. & Valiant, L.G. (1987). Recent results on Boolean concept learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 337–352). Irvine, CA: Morgan Kaufmann.

Kedar-Cabelli, S.T. (1985). Purpose-directed analogy: A summary of current research. *Proceedings of the Third International Workshop on Machine Learning* (pp. 80–83). Skytop, PA: Morgan Kaufmann.

Laird, J.E., Rosenbloom, R.S. & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1,* 11–46.

Larson, J. & Michalski, R.S. (1977). Inductive inference of VL decision rules. *SIGART Newsletter, 63,* 38–44.

Matheus, C.J. (1989). *Feature construction: An analytic framework and an application to decision trees* (Tech. Rep. No. UIUCDCS-R- 89-1559). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S. (1973). Discovering classification rules using variable valued logic system VL1. *Proceedings of the Third International Joint Conference on Artificial Intelligence* (pp. 162–72). Stanford, CA: Morgan Kaufmann.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Michalski, R.S. & Ko, H. (1988). On the nature of explanation or Why did the winebottle shatter? *Proceedings of the 1988 AAAI Spring Symposium Series on Explanation-based Learning* (pp. 12–14). Stanford, CA: AAAI.

Michalski, R.S. & Watanabe, L.M. (1988). *Constructive closed-loop learning: introductory ideas and examples.* (Tech. Rep. No. MLI-Report 88-1). Fairfax, VA: George Mason University, Artificial Intelligence Center.

Mitchell, T.M., Keller, R. & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1,* 11–46.

Muggleton, S. (1987). DUCE, an oracle based approach to constructive induction. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 287–292). Milan, Italy: Morgan Kaufmann.

Pagallo, G. & Haussler, D. (1988). *Feature discovery in empirical learning* (Tech. Rep. No. UCSC-CRL-88-08). Santa Cruz: University of California, Department of Computer Science.

Pazzani, M.J. (1988). *Learning causal relationships: An integration of empirical and explanation based learning methods.* Doctoral dissertation, Department of Computer Science, University of California, Los Angeles.

Rendell, L.A. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 650–658). Los Angeles, CA: Morgan Kaufmann.

Rendell, L.A. (1986). A general framework for induction and a study of selective induction. *Machine Learning, 1,* 177–226.

Sammut, C. & Banerji, R.B. (1986). Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Sterling, L. & Shapiro, E. (1987). *The art of Prolog.* Cambridge, MA: MIT Press.

Utgoff, P.E. (1986). Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM, 27,* 1134–1142.

Watanabe, L. & Elio, R. (1987). Guiding constructive induction for incremental learning from examples. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 293–296). Milan, Italy: Morgan Kaufmann.

Winston, P.H., Binford, T.O., Katz, B. & Lowry, M. (1983). Learning physical descriptions from functional definitions, examples, and precedents. *Proceedings of the Second National Conference on Artificial Intelligence* (pp. 433–439). Washington, DC: Morgan Kaufmann.