Guiding Constructive Induction for Incremental Learning from Examples

Larry Watanabe and Renie Elio

Department of Computing Science, University of Alberta
Edmonton, Alberta T6G 2H1

ABSTRACT

LAIR is a system that incrementally learns conjunctive concept descriptions from positive and negative examples. These concept descriptions are used to create and extend a domain theory that is applied, by means of constructive induction, to later learning tasks. Important issues for constructive induction are *when* to do it and how to *control* it LAIR demonstrates how constructive induction can be controlled by (1) reducing it to simpler operations, (2) constraining the simpler operations to preserve relative correctness, (3) doing deductive inference on an as-needed basis to meet specific information requirements of learning sub-tasks, and (4) constraining the search space by subtask-dependent constraints.

I. INTRODUCTION

In this paper, we describe an incremental learning system called LAIR, which learns conjunctive concept descriptions from positive and negative examples. Incremental learning systems need mechanisms for creating or inducing a concept description from one example, and for modifying that description as additional examples are presented. Several algorithms exist for inducing a concept description that consists of only features and values that appear in the example descriptions [1, 3,6]. The process of introducing new features or descriptors into the concept description, termed constructive generalization, has been formalized as a transformation rule [2]. However, its role in altering the knowledge representation space is not well-understood. Of all the knowledge that could be retrieved or derived from the descriptors that occur in the examples, what is relevant to transforming the current concept description (and transforming it correctly)? Simply put, the unresolved issues about constructive generalization are *when* to do it and how to *guide* it. These are the issues we have tried to address in LAIR.

II. OVERVIEW OF APPROACH

In this section, we outline the general framework and approach we have taken in LAIR. First, we consider "climbing the generalization tree," [6] (which introduces new values for already-seen descriptors) and "constructive generalization" (which introduces new descriptors) as cases of *constructive induction.* Second, we view constructive induction as based on inference and deduction. Third, we use constraints that are inherent in the learning task to limit the knowledge retrieval and deduction. An important result of the second and third points is that LAIR has no "constructive induction" rule among its concept transformation rules. The constructive induction rule is derived from inference rules and concept revision rules that use description constraints. Our approach also uses an "information conservation" principle when generalizing a concept description. Finally, we have a relaxed definition of a "correct" concept description. In LAIR, a transformation of a concept description might make the concept description incorrect with respect to previously-seen (but forgotten) examples, but it is guaranteed to converge on the correct description with enough examples.

A. Conjunctive Concept Descriptions

Descriptions of concepts and examples have form:

$$\lambda x[P_{11}(x) \wedge \ldots \wedge P_{1n}(x) \wedge \neg P_{21}(x) \wedge \ldots \wedge \neg P_{2m}(x)]$$

where each *Pij(x)* is of form $P^n(t1,\ldots\ {}_t t_n)$ where P^n is an n-adic predicate symbol chosen from some fixed, finite set of such symbols, one of the t_i is x, and the other t_i are constants or skolem functions of x. We will refer to the unnegated P_{ij} as Requireds (REQs) and the negated *Py* as NOTs.

Examples are denoted by constants (e.g. Ex-1) and are described by applying a description to the constant:

Xx[Pos(Arch, x) A Block(f(x), x) A Block(g(x), x) A Ontop(f(x),g(x),x)](Ex-l)

"Ex-1 is a positive example of the concept Arch; in Ex-1, there are two blocks, one of which is on top of the other."

One of the predicates in each example description must be *Pos(Concept, x)* if the example is positive, or Pos*(Concept, x)* if the example is negative.

A description *D* is *relatively correct* at time *t* iff for every *remembered* positive example *Ex* of the concept,

KB A *Ex-Description(Ex)* h *D(Ex)*

and for every *remembered* negative example Ex of the concept,

KB A *Ex-Description(Ex)* *D(Ex)*

where *Ex-Description(Ex)* is the description of *Ex,* and *KB* is the knowledge base.** LAIR remembers only one past positive example and the current example, but this definition holds true for any set of remembered examples. Relative correctness is equivalent to correctness in systems that can remember all the examples it has seen at any time.

B. Deductive Inference

LAIR uses the following rules of inference to determine whether constraints on the concept description are satisfied, to classify examples, and to extend example descriptions by adding REQ's and NOTs.

* This work was supported by NSERC Operating Grant A0889 to Renec Elio.

The knowledge base includes predicates and inference rules that relate predicates, and is described in more detail in section III.

1. $P(Ex) \land \ldots \land Q(Ex)] \vdash P(Ex)$
2. $P(Ex) \land \ldots \land \neg Q(Ex) \vdash \neg Q(Ex)$
3. If $KB \land \textit{Ex-Description}(Ex) \vdash P(Ex)$ and
   $KB \land \textit{Ex-Description}(Ex) \vdash Q(Ex)$ then
   $KB \land \textit{Ex-Description}(Ex) \vdash P(Ex) \land Q(Ex)$
4. If $P(x)$ has the form of a REQ, and
   $KB \land \textit{Ex-Description}(Ex) \nvdash P(Ex)$ then
   $KB \land \textit{Ex-Description}(Ex) \vdash \neg P(Ex)$***
5. If $P(x) \Rightarrow Q(x)$ is a rule in the KB, and
   $KB \land \textit{Ex-Description}(Ex) \vdash P(Ex)\sigma$
   then $KB \land \textit{Ex-Description}(Ex) \vdash Q(Ex)\sigma$, where σ is a substitution.

C. Constraints on the Concept Description

LAIR keeps track of whether a predicate is provable or unprovable for some positive example:

Drop(P, *t)* — At or prior to time *t*, *P* was inferred false of some positive example

Some(P, t) — At or prior to time *t*, *P* was inferred true of some positive example.

This knowledge constrains the concept description: concept descriptions cannot include Drop'd predicates as REQs, or Some'd predicates as NOTs.

D- Concept Revision Rules

LAIR uses the following concept revision rules:

- The add-REQ rule:
  If a predicate has never been Drop'd from the concept description
  & it can be proven true of all the remembered positive examples
  Then add the predicate to the concept description
  & remember it is true of Some positive example

- The add-NOT rule:
  If a predicate has not been proven of Some positive examples
  & it can be proven true of the current negative example
  & it cannot be proven true of the remembered past positive example
  Then add its negation to the concept description

- The drop-REQ rule:
  If a predicate in the concept description cannot be proven true of the remembered positive examples
  Then drop it from the concept description
  & remember it has been Drop'd

- The drop-NOT rule:
  If a negated predicate *P* in the concept description can be proven true of any remembered positive example
  Then drop P from the concept description
  & remember it is true of Some positive example

Notice LAIR does not have a constructive induction rule. This rule would be written in our framework as follows:

This rule is based on the assumption that example descriptions are complete with respect to the fixed set of predicates from which descriptions are constructed.

- The constructive induction rule—applies domain knowledge to generalize a concept description.
  If the concept description includes a predicate *P*
  & the knowledge base contains a rule *P* implies *Q*
  Then drop *P* from the concept description
  & add *Q* to the concept description

E. Relative Completeness and Relative Correctness

Two important properties of these rules as implemented in LAIR are "preservation of relative correctness" and "relative completeness." A description revision rule *R preserves relative correctness* iff given description D, if *D* is relatively correct then *R* revises *D* to a description *D'* that is also relatively correct.

Completeness is defined relative to LAIR's ability to use the concept description to classify examples. Suppose there exists a correct description that be proven true of all the positive examples and false of all the negative examples. Then a set of concept revision rules is *relatively complete* iff a correct description *D'* can be derived from every intermediate description *D* derivable from any set of positive and negative examples, using the rules. Since LAIR is designed as a non-backtracking system that does not maintain multiple concept descriptions, these are important properties to ensure that LAIR eventually does find a relatively correct concept description. The constructive induction rule is not relatively complete, so it is not suitable for use in a non-backtracking system that maintains only a single concept description.

The deductive rules and relaxed versions of concept revision rules are equivalent to the constructive induction rule, providing the initial description is relatively correct. The add-REQ rule is relaxed by removing the Drop constraint, and the drop-REQ rule is relaxed by removing all its constraints. The proof in [5] essentially reduces constructive induction to the following steps: use the deduction rule to infer *Q* true of some positive example, use the add-REQ rule to add *Q* to the concept description, and use the drop-REQ rule to drop *P.*

III. LAIR'S KNOWLEDGE BASE

LAIR's knowledge base consists of frames that represent knowledge about examples, concept descriptions, constraints on concept descriptions, and prior or learned knowledge about the domain.

A. Frames Representing Predicates

Knowledge is organized around predicates over examples. Frames corresponding to these entities are created during learning as instantiations of more general predicate frames stored in the prior knowledge base. There are two types of predicate frames: *most-general-predicate* frames and *less-general-predicate* frames. A most-general-predicate frame corresponds to a predicate expression that has more than one argument, e.g., A,x,y,z[body(x, y, z)]. A less-general-predicate frame corresponds to a predicate expression that has only one argument, e.g. A,z[body(a(z), b(a(z)), z)]. Since less-general-predicatcs are the "building blocks" of concept descriptions, constraints on concept descriptions are stored on these frames. The most important slots of a less-general-predicate frame are:

Definition	Lambda expression defining the predicate
Most-general-predicate	the frame representing the most general predicate corresponding to the predicate

Propositions	instances of the predicate
Some	T iff the predicate is true of some positive example
Required	T iff the predicate is an unnegated predicate in th« concept description.
Not	T iff the predicate is a negated predicate in the concept description.
Dropped	T iff the predicate is not derivable for some positive example.
Required-space	T iff the predicate has been considered as a possible REQ in a concept description during the current learning subtask.****
Not-space	T iff the predicate has been considered as a possible NOT in a concept description during the current learning subtask.

The second kind of predicate frame, the most-general-predicate frame, is used to organize knowledge about relationships between predicates and rules. Less-general-predicates can inherit inference rules from most-general-predicates. Most-general-predicates can inherit propositions from less-general-predicates. The most important slots are:

Definition	Lambda expression defining the predicate
Less-general-predicate	Inverse of the most-general-predicate slot on less-general-predicate frames.
Consequent-of	Rules in which the predicate, or one of its less general predicates, is a consequent.
Antecedent-of	Rules in which the predicate, or one of its less general predicates, is an unnegated antecedent
Neg-antecedant-of	Rules in which the predicate, or one of its less general predicates, is a negated antecedent

B. Rule Frames

Rule frames represent knowledge corresponding to logical implications. The most important slots on a rule frame are:

Consequents	Consequents of the implication.
Antecedents	Unnegated antecedents of the implication.
Neg-anteceaents	Negated antecedents of the implication

A typical rule frame might be:

graspable-14	
Consequents	graspable(y, z)
Antecedents	cyl(y, z), small(y, z), light(y, z), body(x, y, z)
Neg-antecedents	hot(y, z)

This rule corresponds to the implication statement If something is light with a small, cylindrical body that is not hot, then it's graspable.

IV. HOW LAIR LEARNS

To describe how LAIR learns, we will explain learning a concept description for "cup" [7]. Assume LAIR has a number of rules relating to the concepts "hot," "stable," "open-vessel," "graspable," and "liftable." LAIR remembers only one previous example, which must be positive, plus the current example.

These subtasks are described in section IV.

The first example is "cha-cup +," so LAIR forms the following description as the past positive example: "A positive example of a cup is something with a flat bottom, an upwards-pointing concavity, a small, cylindrical body, that was lifted."

Since initially there are no constraints on the concept description, all of the features can be added to the concept description, which was initially empty. The second example, "typical-cup +," has the description: "A positive example of a cup is something with a flat bottom, an upwards-pointing concavity, a small, cylindrical body, that is light and has a handle." The concept description is revised to include the new features "handle" and "light"

Relative correctness is checked by trying to prove the extended concept description true of the remembered positive examples. LAIR restores relative correctness by dropping the "was-lifted" predicate (not true of example 2) and the "handle" and "light" predicates (not true of the remembered past positive example).

Although the concept description is relatively correct, in some sense information has been "lost." LAIR attempts to conserve the lost information by specializing the revised concept description with inferences it can make from the dropped REQ's. Essentially, this means "Docs this descriptor, which I've decided to drop, imply something else that is provable of both the current example and the past positive example(s)?" Information conservation is important in LAIR for several reasons. First, when the current example is positive, then LAIR is essentially learning from positive-only examples. Systems that learn from positive-only examples must have methods to avoid over-generalization. Information conservation avoids over-generalization by compensating for the loss of information in generalization by a specialization step. Further, specialization is focused only on descriptors that can be inferred from the dropped REQ's.

This information conservation involves searches for inferrable descriptors by looking at rule-generalizations of the dropped REQ's. A predicate P is a rule-generalization of a predicate Q if (1) Q has a Most-gen-pred that is an Antecedent-of a rule that has P as a Consequent, or (2) P is a rule-generalization of another predicate that is a rule-generalization of Q. The relationship between P and Q is shown below.

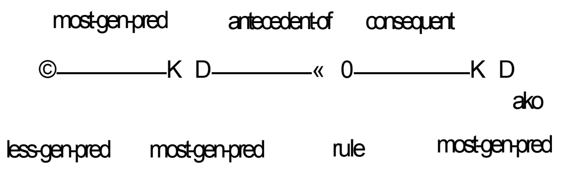

Figure 1. Rule Generalizations
i

A subtask is created for each dropped REQ by initializing a "REQ-boundary" to the dropped REQ. Each subtask is an heuristic search of the space of descriptors to find a less general predicate of a rule-generalization of the dropped REQ that can be add-REQ'd to the concept description. The subtask chooses a REQ-boundary element P and tries to add-REQ P (or a less general predicate than P). If this succeeds, then the subtask terminates successfully. Otherwise, P is deleted from the REQ-boundary and P's rule-generalizations are added to the REQ-boundary if P (or a less general predicate than P) satisfies the constraint that it is true of a remembered, positive example of the concept, and is not currently a REQ. If the REQ-boundary is empty, then the subtask terminates unsuccessfully, otherwise a new REQ-boundary element is chosen and the process is repeated.

For the "typical-cup +" example, the first REQ-boundary is initialized to {"was-lifted"}. Since "was-lifted" was dropped, it cannot be add-REQ'd, and is removed from the REQ-boundary. However, "was-lifted" has a rule-generalization: If an object was-lifted, then it is lift able.

Since "was-lifted" satisfies the REQ-boundary constraint, its rule-generalization can be added to the REQ-boundary. LAIR proves that "liftable" is true of the current example (Ex-2) and the past positive example (Ex-1) by accessing and applying inference rules such as: If something is graspable and light, then it is liftable; If something has a handle then it is graspable.

Therefore, "liftable" is added to the concept description. REQ-boundaries arc also created for "handle" and "light," but no further information can be conserved for these cases. The resulting, relatively correct concept description is: "Something that is liftable, with a flat bottom, upwards pointing concavity, and a small, cylindrical body."

As one further positive example, consider "balanced-cup +," whose description is: "A positive example of a cup is something that is balanced, contains something, has a handle, and is light." LAIR revises the concept description, and conserves information as for the "typical-cup +" example, resulting in the desired concept description: "Something that is liftable, stable, an open-vessel, and has a body." Note that the "balanced-cup +" example differed from the concept description in several ways. This allows LAIR to drop more irrelevant features and add more relevant features. Using information conservation, LAIR learns faster when positive examples show the typical variance in the concept, i.e., when "far-hits" are presented. Second, "body" is not eliminated from the concept description, although it is not necessary for relative correctness. LAIR's goal is just to find a correct description; demanding "minimally" correct descriptions is beyond its scope.

LAIR learns concept descriptions with negated predicates using a method similar to the one outlined above. Assume the current concept description is: "Something with a small, cylindrical body." The past positive example is "insulated-object +," whose description is: "A positive example of 'graspable' is something insulated with a small, cylindrical body, and hot contents." The current negative example is "uninsulated-object -," whose description is: "A negative example of 'graspable' is something with a small, cylindrical body, and hot contents." The concept description incorrectly classifies this example as positive, so the concept description is specialized by an "add-REQ subtask" (with a REQ-boundary) and an "add-NOT subtask" (with a NOT-boundary). Neg-rule-generalizations are used by this task. A predicate P is a neg-rule-generalization of a predicate Q iff Q has a Most-gen-pred that is a Neg-antecedent-of a rule that has P as a Consequent.

The add-REQ subtask initializes the REQ-boundary to all predicates true of the past positive example but not of the current negative example. Next, the add-REQ subtask chooses a REQ-boundary element Q and tries to add-REQ it to the concept description. If the attempt is successful, then the Differencing Task terminates. Otherwise, Q is deleted from the REQ-boundary. If Q is true of the Past example, false of the Curr example, and is not currently a REQ'd, then its rule-generalizations are added to the REQ-boundary, and its neg-rule-generalizations are added to the NOT-boundary.

The add-NOT subtask initializes the NOT-boundary to all predicates true of Curr but not of Past. Next the add-NOT subtask chooses a NOT-boundary element Q and tries to add-NOT it to the concept description. If the attempt is successful, then the Differencing Task terminates. Otherwise, Q is deleted from the NOT-boundary. If Q is true of the Curr example, false of the Past example, and is not currently a NOT, then its rule-generalizations are added to the NOT-boundary, and its neg-rule-generalizations are added to the REQ-boundary.

LAIR alternates between these two subtasks, since each adds new elements to the other's boundary. Success by either the add-REQ subtask or the add-NOT subtask produces a relatively correct description.

V. IMPLEMENTATION DETAILS

LAIR is implemented as a frame and rule-based system in InterLisp-D and OPS4 running on a Xerox 1186 workstation. LAIR learns the domain theory for the "cup" domain, consisting of 1 rule for "hot" and "cup," 2 rules each for "stable," "open-vessel," graspable," and "liftable." Learning requires 26 examples and takes about 570 seconds of CPU time.

VI. CONCLUSIONS

LAIR makes some prelimary steps towards suggesting how constructive induction can be guided. First, constructive induction can be decomposed to simpler operations. Second, the simpler operations can be constrained so that they preserve relative correctness and are relatively complete. Third, inference need not be done all at once upon presentation of an example, but on an as-needed basis to meet the specific needs of the learning process at each point in time. This contrasts with a related approach in MARVIN [4] that computes the logical closure of each presented or generated example as a preliminary step to revising the concept description. Fourth, constraints can be formulated on how the space of descriptors is searched by specifying boundary conditions for specific learning subtasks.

LAIR can be compared to explanation-based methods for learning concepts such as "cup." Explanation-based systems [7] require that the domain knowledge and the goal concept be given to the system whereas LAIR induces both these kinds of knowledge from the examples. However, LAIR may not learn the best description for efficient recognition of instances of the concept. Explanation-based systems try to induce efficient descriptions by imposing an operationality criterion [8] on learned descriptions.

REFERENCES

[1] Dietterich, T. G. and Michalski, R. S. (1983). A comparative review of selected methods for learning structural descriptions. In Michalski, R. S., Carbonell, J. G.and Mitchell, T. M. (Eds.). Machine Learning: An Artificial Intelligence Approach, Vol I. Tioga: Palo Alto.

[2] Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol I. Tioga: Palo Alto.

[3] Mitchell, T. M. (1977). Version spaces: a candidate elimination approach to rule learning, Fifth International Joint Conference on Artificial Intelligence, 305-310.

[4] Sammut, C, and Banerji, R. B. (1986). Learning concepts by asking questions. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol II. Tioga: Palo Alto.

[5] Watanabe, L. (1987). Guiding Constructive Induction. M.Sc. Thesis, University of Alberta.

[6] Winston, P. H. (1975). Learning structural descriptions from examples. In Winston, P. H. (Ed.), The Psychology of Computer Vision, McGraw Hill: New York.

[7] Winston, P. H., Binford, T. O., Kate, B. and Lowry, M. (1983). Learning physical descriptions from functional definitions, examples, and precedents. Proc. AAAI-83, 433-439.

[8] Mitchell, T. M., Keller, R. and Kedar-Cavelli, S. (1986). Explanation-based generalization: a unifying view, Machine Learning 1, No. 1, January.