

Concept Learning Using EBG As an Abstraction Mechanism

William Cohen

Department of Computer Science
Rutgers University

Outline

- Motivation
- Basic ELGIN algorithm
- Analysis of ELGIN
- Experimental results
- Extensions to ELGIN
 - Approximations
 - Incremental learning
 - Generalization-to- N
- Summary and Conclusions

Background knowledge in concept learning

A concept learner

- inputs examples of an unknown *target concept*
- outputs a description of the target concept

Concept learning is difficult unless the set of possible target concepts is very constrained. This thesis is based on the following premise:

The reason that concept learning seems difficult is that most concept learning systems make use of only a limited amount of “background knowledge” about the concept to be learned.

Background knowledge in concept learning

The thesis investigates techniques that

- input
 - examples of the *target concept*
 - background knowledge encoded as a *theory*
- output a description of the target concept

In other words, techniques that map

$$Theory + Examples \rightarrow Hypothesis$$

rather than

$$Examples \rightarrow Hypothesis$$

Concept learning and EBG

An existing technique that does this is EBG (!)

Normally, EBG performs the function

$$Domain-Theory + Example \rightarrow Rule$$

where $(Domain-Theory \models Rule)$. Suppose one

- Applies EBG to each positive example
- Treats the resulting set of rules as a new theory, distinct from the domain theory.

Then EBG is performing the function

$$\begin{array}{ccccc} Domain-Theory & + & Examples & \rightarrow & New-Theory \\ || & & || & & || \\ Theory & + & Examples & \rightarrow & Hypothesis \end{array}$$

Concept learning and EBG

An example of such a system is LEAP. In LEAP

Domain-Theory \equiv *Correct-Circuit-Transformation*

Target-Theory \equiv ‘*Useful*’-*Circuit-Transformation*

LEAP succeeds when *New-Theory* = *Target-Theory*

Notice that (*Domain-Theory* \models *Target-Theory*), but (*Target-Theory* $\not\models$ *Domain-Theory*); the target theory is a *specialization* of the domain theory.

When *Domain-Theory* \neq *Target-Theory*, there are some proofs in the domain theory of goals that are not true in the target theory. In other words: there are some *explanations*^a that are “wrong_T”.

^aAn “explanation” is a proof in the domain theory

Assumptions about the domain theory

What does LEAP assume about *Domain-Theory*?

What should one assume about *Domain-Theory*?

- Every positive example has one “right_{*T*}” explanation, and applying EBG to the “right_{*T*}” explanation produces a rule that is part of *Target-Theory*.

What about “wrong_{*T*} explanations”?

- Any example may have any number of “wrong_{*T*}” explanations: in particular,
 - A positive example may have several “wrong_{*T*}” explanations as well as the “right_{*T*}” one.
 - A negative example may have any number of “wrong_{*T*}” explanations.

Assumptions about the domain theory

Why are these assumptions reasonable?

1. There are interesting classes of theories that satisfy these conditions (for example, theories that use *assumptions* as well as facts in proofs).
2. Experimental evidence.

Theories that satisfy these conditions will be called *abductive* theories.

The main contribution of the thesis is developing algorithms that learn from examples and ‘abductive’ theories.

Outline

- Motivation
- Basic ELGIN algorithm
- Analysis of ELGIN
- Experimental results
- Extensions to ELGIN
 - Approximations
 - Incremental learning
 - Generalization-to- N
- Summary and Conclusions

The ELGIN algorithm

Algorithm ELGIN($S^+, S^-, \Theta, \mathcal{O}$):

1. Compute the explanation structure (in Θ) of every proof of every example in S^+ .
2. For each explanation structure found in step 1, find the set of candidate rules that can be formed by
 - (a) modifying the explanation structure by marking some constrained set of internal nodes as “operational”
 - (b) applying the final stage of EBG (using operationality predicate \mathcal{O}) to the resulting explanation structure to form a rule.
3. Filter the set of candidate rules by removing any rule that covers an element of S^- .
4. Use a greedy set cover to find a small set of rules that accounts for all of the training examples.
5. Return the covering set of rules as the hypothesis.

Generalization: k -complete

How does one choose what constrained set of internal nodes to mark as “operational” ?































One possibility: choose any set of k nodes (for some fixed k) and mark them as operational. The class of rules obtained this way are *k -complete* rules.

An example: a domain theory




- r1: $\text{bid}(H, S1) \Leftarrow$
 $\text{opening_strength}(H) \wedge \text{biddable}(H, S1)$
 $\wedge \text{biddable}(H, S2) \wedge \text{prefer}(H, S1, S2)$
- r2: $\text{opening_strength}(H) \Leftarrow * \text{strong_opener}(H)$
- r3: $\text{opening_strength}(H) \Leftarrow * \text{weak_opener}(H)$
- r4: $\text{biddable}(H, S) \Leftarrow * 6 \text{cards}(H, S)$
- r5: $\text{biddable}(H, S) \Leftarrow * 5 \text{cards}(H, S)$
- r6: $\text{biddable}(H, S) \Leftarrow * 4 \text{cards}(H, S)$
- r7: $\text{prefer}(H, S1, S2) \Leftarrow * \text{longer}(H, S1, S2)$
- r8: $\text{prefer}(H, S1, S2) \Leftarrow * \text{higher}(S1, S2)$
 (in ordering $\clubsuit, \diamondsuit, \heartsuit, \spadesuit$)

* This predicate is operational according to \mathcal{O} .

An example: the training data

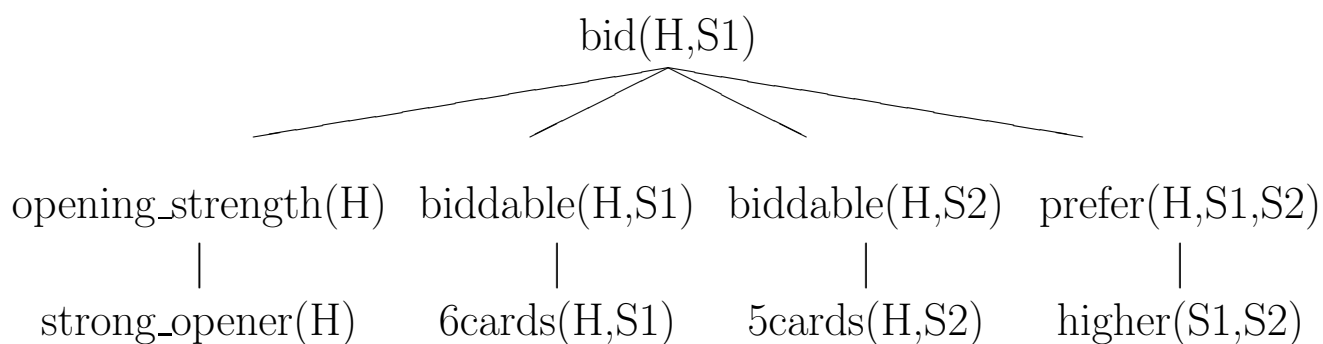
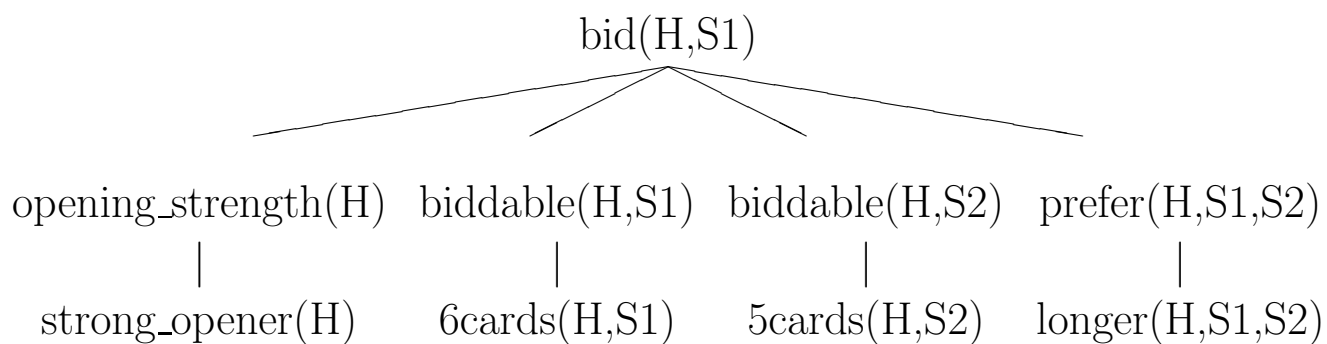
	Training Example	+/-
1a	bid( AQJ987  K2  KQJ74  —, )	+
1b	bid( AQJ987  K2  KQJ74  —, )	—
2a	bid( AQ987  32  KQJ742  —, )	+
2b	bid( AQ987  32  KQJ742  —, )	—
3	bid( AQ98  32  KQJ742  9, )	—
4	bid( AQJ97  K2  KQJ874  —, )	—

Notice that:

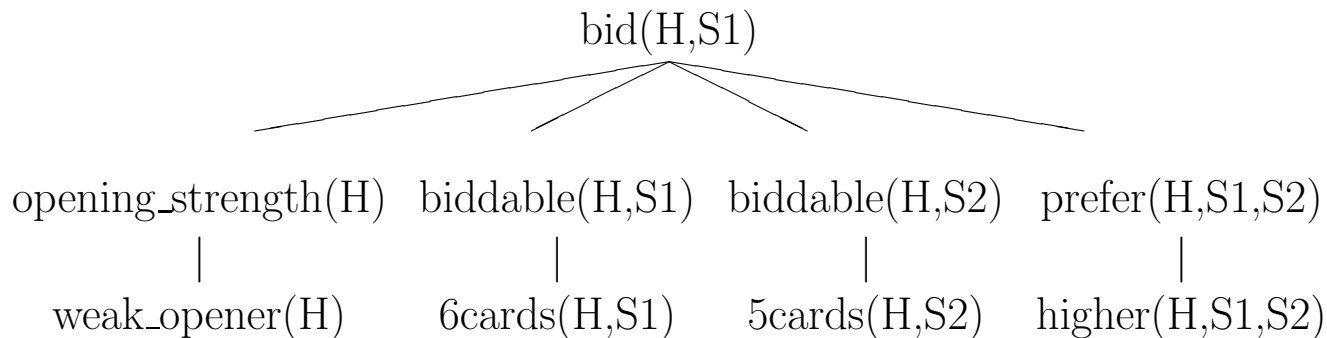
- some incorrect bids have explanations in T_i .
(T_i suggests  for 2b since  suit is longer.)
- some correct bids have multiple explanations in T_i .
(T_i suggests  for 1a because spades are the longer suit AND because spades are the higher suit.)

An example: the set of candidate rules

For “bid(**♠** AQJ987 **♥** K2 **♦** KQJ74 **♣** —, **♠**)”:



For “bid(**♠** AQ987 **♥** 32 **♦** KQJ742 **♣** —, **♠**)”:



An example: the set of candidate rules

The following rules can be formed from the preceding explanation structures:

$$\text{strong_opener}(H) \wedge 6\text{cards}(H,S1) \wedge \\ 5\text{cards}(H,S2) \wedge \text{longer}(H,S1,S2) \Rightarrow \text{bid}(H,S1)$$
$$\text{opening_strength}(H) \wedge 6\text{cards}(H,S1) \wedge \\ 5\text{cards}(H,S2) \wedge \text{longer}(H,S1,S2) \Rightarrow \text{bid}(H,S1)$$
$$\text{strong_opener}(H) \wedge \text{biddable}(H,S1) \wedge \\ 5\text{cards}(H,S2) \wedge \text{longer}(H,S1,S2) \Rightarrow \text{bid}(H,S1)$$
$$\text{strong_opener}(H) \wedge 6\text{cards}(H,S1) \wedge \\ \text{biddable}(H,S2) \wedge \text{longer}(H,S1,S2) \Rightarrow \text{bid}(H,S1)$$
$$\text{strong_opener}(H) \wedge 6\text{cards}(H,S1) \wedge \\ 5\text{cards}(H,S2) \wedge \text{prefer}(H,S1,S2) \Rightarrow \text{bid}(H,S1)$$

⋮

An example: the hypothesis

In this case, the set cover routine would output the following set of rules as T_s :

$\begin{aligned} &\text{strong_opener}(H) \wedge 6\text{cards}(H,S1) \wedge \\ &\text{biddable}(H,S2) \wedge \text{longer}(H,S1,S2) \Rightarrow \text{bid}(H,S1) \\ \\ &\text{weak_opener}(H) \wedge \text{biddable}(H,S1) \wedge \\ &5\text{cards}(H,S2) \wedge \text{higher}(H,S1,S2) \Rightarrow \text{bid}(H,S1) \end{aligned}$

Outline

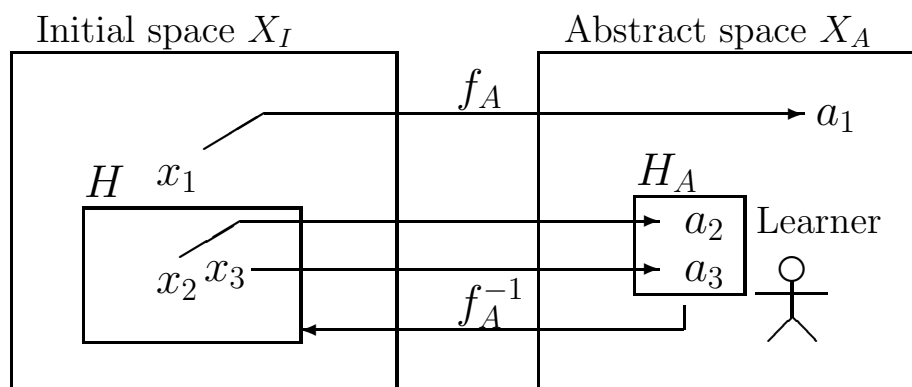
- Motivation
- Basic ELGIN algorithm
- Analysis of ELGIN
- Experimental results
- Extensions to ELGIN
 - Approximations
 - Incremental learning
 - Generalization-to- N
- Summary and Conclusions

Alternative view: EBL as abstraction

Recall the LEAP system.

An alternative way of viewing such a system is as an “abstraction-based learning system”, where

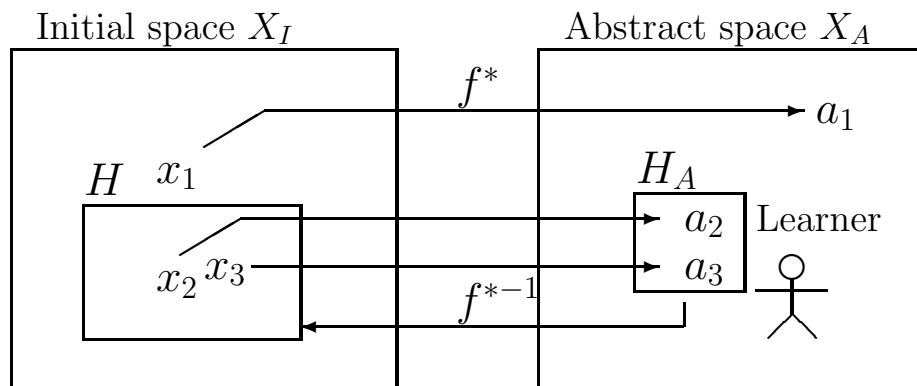
- f_A maps examples to explanation structures (the first stage of EBG)
- f_A^{-1} maps (sets of) explanation structures to (sets of) rules (the second stage of EBG)
- The learner is simply rote learning.



Other abstraction-based learning systems include MetaDEN-DRAL and IOE.

ELGIN as abstraction

ELGIN is actually implemented as an abstraction-based learning system, where f_A (roughly) maps examples to sets of explanation structures

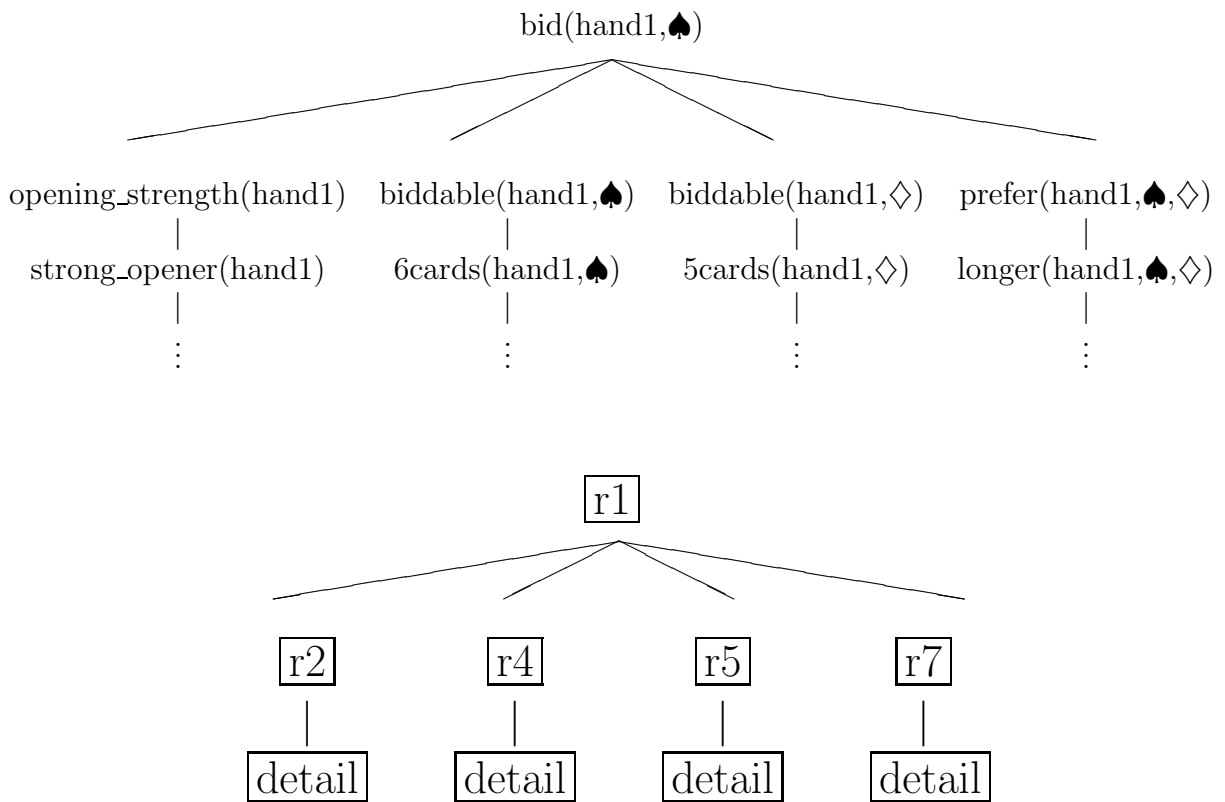


The next few slides describe more fully

- The abstraction function f^*
- The abstract hypotheses
- How to invert the abstraction function

A clause name tree

A *clause name tree* is formed by replacing each node of the proof tree with the name of the clause used to prove that goal, and replacing proofs of “details” with detail.



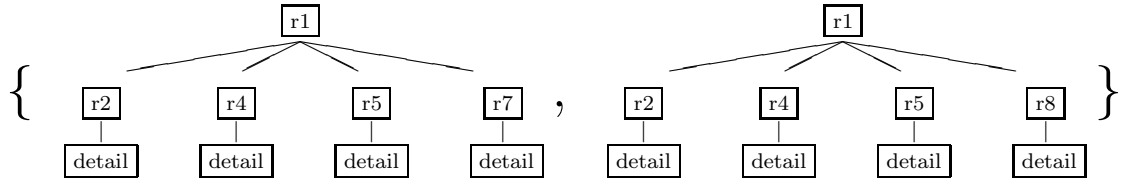
ELGIN's abstraction function

The *abstraction function* f_A is

$$f_A(x) \equiv \{ e : e \text{ is the clause name tree} \\ \text{for some proof of } x \}$$

For instance,

$$f_A(\text{bid}(\spadesuit \text{ AQJ987 } \heartsuit \text{ K2 } \diamondsuit \text{ KQJ74 } \clubsuit -, \spadesuit)) \equiv$$



Appropriate abstraction functions

APPROP (f_A, \mathcal{P}) denotes the proposition: “the abstraction function f_A is ‘appropriate’ for all target concepts T in \mathcal{P} ”.

Definition:

$$APPROP(f_R, \mathcal{P}) \equiv \exists T_A \subseteq range(f) : T = f^{-1}(T_A)$$

I.e. an ideal learner can still identify T via a hypothesis in the abstract space.

Abstraction function: Analysis

Theorem: For the function f_A used by ELGIN,

$$APPROP(f_A, \mathcal{P}) \Leftarrow$$

$$\forall T \in \mathcal{P}, \forall x \in T,$$

$\exists p_x$ that is a proof of x :

EBG applied to the proof p_x returns
a generalization that is a subset of T

I.e. Use of f_A is justified by a *weak correctness assumption about the domain theory*.

This is one of several results obtained via a general technique for analyzing abstraction functions for concept learning. (Cohen, ML90)

PAC Analysis

Definition: Let all examples be chosen according to a probability distribution function D , and let the (representation of) the target T concept have size n .

A concept learning algorithm $LEARN$ is a *pac-learning* algorithm with sample complexity m when for every such D and T , if at least $m(1/\epsilon, 1/\delta, n)$ examples are given to $LEARN$ then

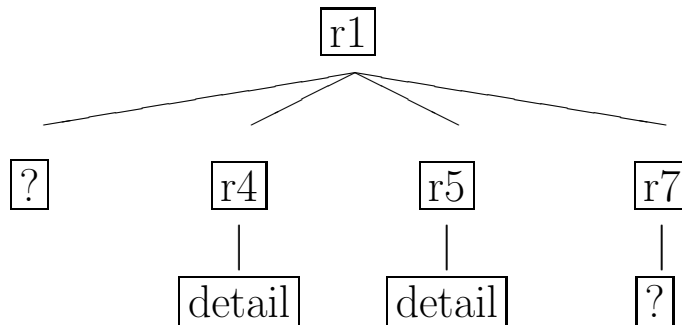
$$Prob(\text{error of } LEARN\text{'s hypothesis} > \epsilon) < \delta$$

I.e. , $LEARN$ (probably) produces an (almost) correct hypothesis using m examples.

This sort of analysis is often based on counting the number of possible hypotheses of size bounded by n .

Hypotheses in the abstract space

A *clause name tree prefix* contains “wildcards” that match any subtree, and is interpreted as the set of clause name trees that match it.



Remember the “hitting set” problem?

Some sets:

$$S_1 = \{a, c\}$$

$$S_2 = \{b, c, d\}$$

$$S_3 = \{b, d\}$$

Some “hitting sets”
of S_1, S_2, S_3 :

$$\{c, d\}$$

$$\{a, b\}$$

The hitting set problem: find a minimal set of elements that “hits” (intersects with) each S_i .

New terminology: the set of all sets that could be hit by a hitting set h is the *hitset* of h .

Hypotheses in the abstract space

ELGIN uses unions hitsets of prefixes as hypotheses

$$hitset(P_1) \cup \dots \cup hitset(P_k)$$

As it turns out, applying the following procedure to a prefix P computes $f_A^{-1}(hitset(P))$.

1. Find the explanation structure that corresponds to P , treating nodes labeled $\boxed{?}$ exactly as nodes labeled $\boxed{\text{detail}}$ are usually treated.
2. Apply the final stage of EBG to form a rule from this explanation structure.

PAC Analysis

- ANA-EBL is a pac-learning algorithm with sample complexity

$$m(\frac{1}{\epsilon}, \frac{1}{\delta}, n) = O(\max(\frac{1}{\epsilon} \log \frac{1}{\delta}, \frac{n \log |\Theta|}{\epsilon} (\log \frac{n \log |\Theta|}{\epsilon})^2))$$

where $|\Theta|$ is the number of clauses in the domain theory Θ , and n is the total size of the rules in the target theory.

- Modulo the cost of applying f_A , ELGIN runs in time polynomial in the total size of the explanation structures for elements of S^+ but exponential in k .

Outline

- Motivation
- Basic ELGIN system
- Analysis of ELGIN
- Experimental results
- Extensions to ELGIN
 - Approximations
 - Incremental learning
 - Generalization-to- N
- Summary and Conclusions

Application: bridge bidding

Application: Learning opening bids in the game of contract bridge from bidding rules in a textbook and examples.

The domain theory:

1. Translated first three chapters of Sheinwold's "5 Weeks to Winning Bridge" into a Horn clause domain theory.
2. Theory has two flaws:
 - (a) some bidding rules are merely heuristics
 - (b) some predicates are undefined

Bridge bidding: heuristic rules

In the textbook, *examples are used to clarify ambiguity in the text* such as heuristic bidding rules.

Sheinwold, page 16: “[With two biddable suits] the general rule is, if your suits are unequal in length, bid the longer one; if your suits are equal in length, bid the higher one ... [But] you have to disregard this general rule on some hands”.

Fifteen examples then follow which clarify the rule.

Sheinwold, page 23: “[With three biddable suits], as a rule, start with lowest or middle suit ... The examples show how easy it is to keep the bidding low”.

Four examples follow.

Bridge bidding: undefined terms

Sheinwold, page 13-14: “Any hand with 10-11 high card points, 2 or more quick tricks, a comfortable rebid and *good length in the major suits* [hearts and spades] can be opened.”

Common sense constraints give a “partial definition” of *good length in the major suits*: having length in the major suits greater than L for some unknown minimal length L .

So, add to the theory all of these axioms

minlength(1)
minlength(2)
:
minlength(13)

Result: multiple inconsistent explanations of why “good length in the major suits” is true, *one of which is correct*.

Bridge bidding: experiments on fixed data

Using the examples from the text as training examples:

1. Using ELGIN with 0-complete prefixes, the predicate *opening_strength* was learned from a theory manually extracted from the text.
2. The learned definition of *opening_strength* was spliced into the original theory, and ELGIN was run using this theory and k -complete prefixes for $k = 0$, $k = 1$ and $k = 2$.

Each of these theories were then tested on the relevant questions from the sample test from the text.

Theory	Accuracy
Textbook theory	12/16
Textbook theory with learned <i>opening_strength</i>	12/16
output of ELGIN ($k = 0$)	14/16
output of ELGIN ($k = 1$)	15/16
output of ELGIN ($k = 2$)	15/16

Bridge bidding: experiments on random data

For a second set of experiments, a program was written that

- randomly generates bridge hands and
- bids them using Sheinwold's system

The procedure has advantages and disadvantages.

- Unlimited amounts of test data are available, BUT
- an additional source of bias has been introduced.

Experimental results

[opening-strength - from thesis]

Experimental results

[opening-bid - from thesis]

Other concept learning applications

- Using theory of possible analogies, ELGIN learns rules of variable assignment in BASIC [Burstein ML2].
- Using very weak theories indicating only what features are relevant, ELGIN learns various relational concepts including
 - Winston’s “arch” concept
 - Michalski’s “eastbound train” concept
 - Sequence extrapolation rules for Eleusis

Application: control rules

Application: Learning control rules for Prolog programs.

Motivation:

- Important problem (lots of AI problems are easy to formulate as search problems)
- A traditional use of EBL is learning control for search programs.
- Good benchmark problem for a concept learner
- Prolog was chosen as a search program which
 - has simple semantics
 - powerful
 - efficient
 - readily available

Application: control rules

- Rules learned are *clause selection rules*; *i.e.*, the concept being learned is

$$\text{useful}(\text{Clause}, \text{Goal})$$

- Domain theory contains clauses of the program and also an axiom of the form

$$\text{useful}(c_i, A) \leftarrow B_1, \dots, B_q$$

for each clause $c_i : (A \leftarrow B_1, \dots, B_q)$ in the Prolog program.

- Examples are provided by a *critic* and *compiled* into a new program.

Control rules: comparison to EBL

The “standard EBL control learning algorithm” incrementally adds control rules whenever a decision is made for which no control rule exists.

The convergence rate of ELGIN with k -complete prefixes and $k = 0$ was compared to a “standard EBL” on several domains:

- Keller’s MetaLEX (symbolic integration)
- Robot world planner
- Blocks world planner
- Graph search

The quantity measured was how often the learned program (with deterministic control) succeeded on a set of benchmark problems.

Control rules: comparison to EBL

Domain	Accuracy on Benchmark	
	std EBL	ELGIN
μ LEX	100.0	100.0
RW	93.4	98.6
BW	63.9	98.6
GRID	90.2	89.5
Average	86.6	96.7

Summary of control rule experiments

Control rules: comparison to EBL

[BW performance curves]

Outline

- Motivation
- Basic ELGIN system
- Analysis of ELGIN
- Experimental results
- Extensions to ELGIN
 - Approximations
 - Incremental learning
 - Generalization-to- N
- Summary and Conclusions

Analysis of swamping

ELGIN learns faster than EBL, but the control rules it learns are sometimes slower.

Why?

The *utility problem* in EBL is that rules with high *match cost* and low coverage may degrade performance. To solve the utility problem one must *simultaneously*

- maximize coverage
- minimize cost

ELGIN, however, maximizes coverage without regard to cost.

Maximizing utility

A standard technique for multiple-resource optimization problems is

- set an artificial bound on one resource
- maximize the other resource subject to that bound

In learning control rules, constrain ELGIN to only consider control rules with match cost below a certain fixed cutoff.

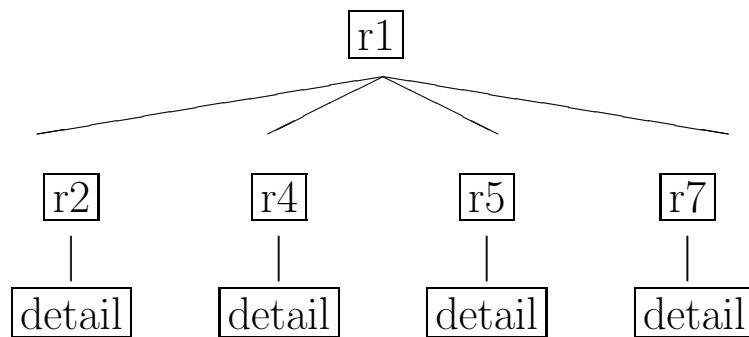
How?

Allow ELGIN to consider *approximate* rules of bounded length. This can be done by changing the class of prefixes.

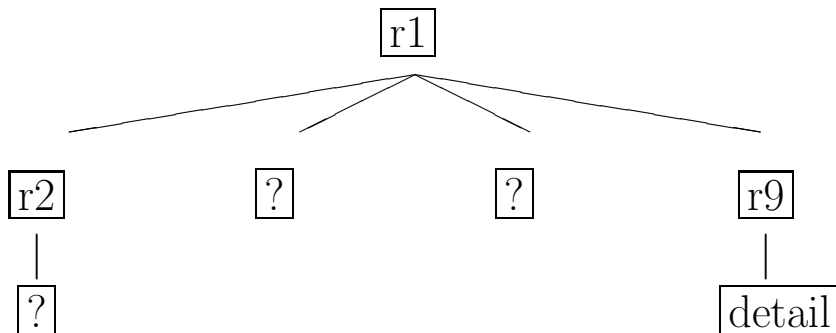
An alternative to k -complete rules

The set of k -tip prefixes which cover a clause name tree e is constructed by picking all sets of k nodes in e and replacing every node which is not a direct ancestor of one of them with $[?]$.

For example,



is covered by



Choosing a set of prefixes

To approximate: when computing the rule that corresponds to a prefix, *discard* the conjuncts associated with wildcard nodes.

Effect: if the class of k -tip rules is used, then control rules contain no more than k conjuncts.

This means that match cost is $O(n^k)$, given the restriction that each “detail” have $O(n)$ proofs (where n is problem size).

Results

Domain	Average Time on Test Problems				<u>ELGIN2</u>
	Nolearn	EBL	ELGIN1	ELGIN2	EBL
μ LEX	2.019	0.103	0.103	0.104	1.01
RW	9.956	2.768	0.922	1.596	0.58
BW	3.150	2.553	4.774	0.987	0.39
GRID	0.137	0.361	0.423	0.143	0.40
Average	4.320	1.447	1.555	0.708	0.49

Summary of experimental results (CPU time)

Domain	Percent Improvement		
	EBL	ELGIN1	ELGIN2
μ LEX	95.9	95.9	94.9
RW	72.2	90.7	84.0
BW	19.0	-51.6	68.7
GRID	-163.5	-208.8	-4.4
Average	18.4	-5.9	60.8

Summary of experimental results
(percent improvement)

An incremental version of ELGIN

One problem with ELGIN is that it is not incremental. iELGIN is an incremental version of ELGIN.

Algorithm iELGIN($x, \Theta, Label$):

1. Find all candidate rules R derivable from some explanation in Θ of x .
2. Predict “true” (*i.e.*, $x \in Target-Theory$) iff

$$\sum_R \frac{weight(R)}{2^{size(R)}}$$

3. If $Prediction = true$ and $Label = false$ then for each R , set $weight(R) \leftarrow 0$.
4. If $Prediction = false$ and $Label = true$ then for each R , set $weight(R) \leftarrow 2 * weight(R)$.

(Weights are initially all equal to one).

Analysis

- iELGIN makes only $O(n)$ wrong predictions en route to learning to predict a target concept of size n .
- iELGIN's predictions probably become accurate after m examples, where

$$m(\frac{1}{\epsilon}, \frac{1}{\delta}, n) = O(\frac{n \log |\Theta|}{\epsilon} \log \frac{1}{\delta} + \frac{n \log |\Theta|}{\epsilon} \log(n \log |\Theta|))$$

iELGIN was also evaluated experimentally.

Experimental results

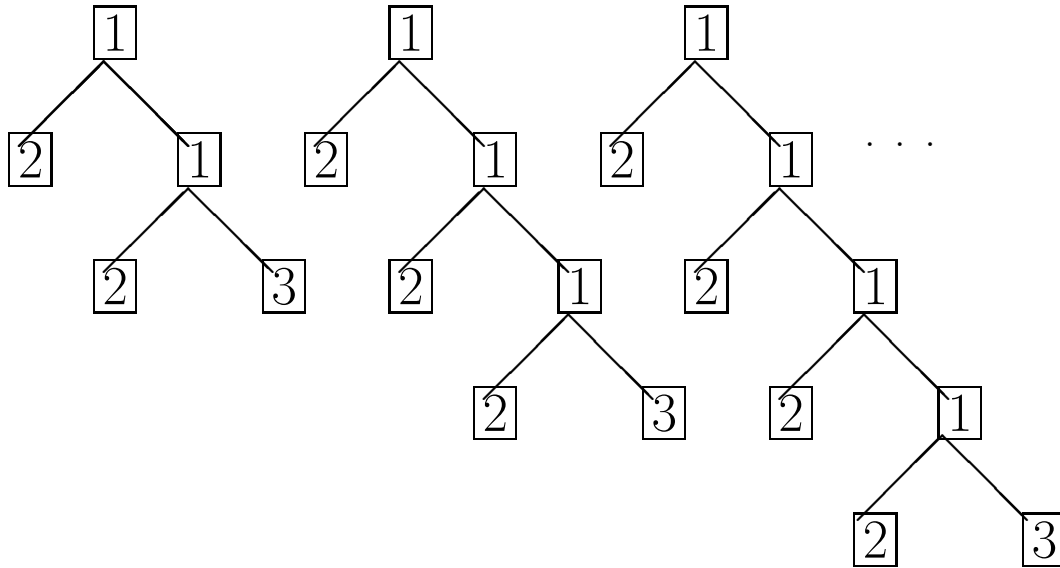
[slides for $k=0$]

Experimental results

[slides for $k=1$]

Generalization-to- N in ELGIN

ELGIN is unable to express an important class of generalization: generalization of the “iterative structure” of an explanation. For example, there is no prefix equivalent to the set



Such sets of name trees naturally arise when the domain theory is recursive.

Generalization-to- N in ELGIN

For example, from the training example
implement_by(AND(OR(a, b), OR(c, d)), NOR(NOR(a, b), NOR(c, d)))
LEAP forms the generalization
implement_by(AND($F1$, $F2$), NOR(NOT($F1$), NOT($F2$)))
where $F1$ and $F2$ can be any boolean functions, but cannot generalize to the N -input case.

Generalization-to- N in ELGIN

How can generalization-to- N be added to ELGIN?

1. Find a scheme for *representing* iteration in trees.

ELGIN uses deterministic tree automata, a generalization of DFAs that accept a set of trees (rather than a set of strings).

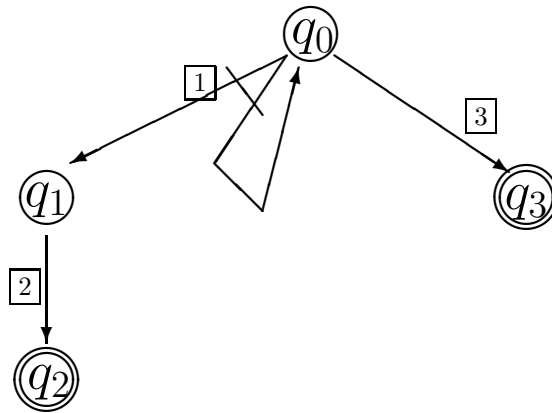
2. Find a technique for *detecting* iteration in trees.

An appropriate technique is described in an appendix of the thesis.

3. Integrate the technique with ELGIN.

Deterministic tree automata

Intuitively, a tree automaton can be thought of as a finite automaton which reads in, not a string, but a tree. Just as DFAs can be represented by a directed graph with labeled edges, DTAs can be represented as a directed *hypergraph*.



Loop induction for trees

Iteration is detected by a “loop induction algorithm.”

Definitions.

- A *loop induction algorithm for trees* is any algorithm that takes as input a name tree t and outputs a DTA A such that $t \in L(A)$.
- A loop induction algorithm is *conservative* wrt \mathcal{P} if it does not overgeneralize t for any $t \in T \wedge T \in \mathcal{P}$
- A loop induction algorithm is *c-concise* wrt \mathcal{P} if for all $T \in \mathcal{P}$, the sum of the sizes of all possible outputs given some $t \in T$ is bounded by $c * size(T)$:

$$\left(\sum_{A \in LI(T)} size(A) \right) \leq c \times size(T)$$

Extended version of ELGIN

The following extension of ELGIN allows it to generalize number using an arbitrary c -concise conservative loop induction algorithm LI .

Algorithm ELGIN($S^+, S^-, \Theta, \mathcal{O}$):

1. Compute a set of candidate rules as follows:
 - Compute the clause name tree (in Θ) of every proof of every example in S^+ .
 - Apply LI to each tree to produce a DTA.
2. Filter this set of DTAs using S^- .
3. Perform the greedy set cover as before, using the set of DTAs remaining after step 2 as the set of candidate rules.

The thesis also considers (formally) the possibility of integrating loop induction with use of prefixes, and discusses other issues (*e.g.*, how to convert a DTA into a rule.)

Results

ELGIN (using a particular loop induction algorithm) has been applied to twelve small learning problems that require generalization-to- N , taken from (Cohen, IMLC88; Shavlik, thesis; Shavlik, IJCAI-89). In each case, it find the intuitively “correct” generalization.

This version of ELGIN is especially useful when an example has multiple explanations, some of which have a clear iterative structure and some of which do not. *ELGIN can look at the alternative explanations and discover which one leads to the best generalization.* The thesis describes a slight variant of a problem from (Shavlik, thesis) that has this property.

Summary

- ELGIN, a theory-guided learning system that can be thought of as using abstraction to make use of background knowledge.
 - Clean separation between learning and applying background knowledge
 - Novel analysis of effect of abstraction
 - Standard analysis of learning techniques
- ELGIN uses a novel abstraction function.
- ELGIN uses a novel learning mechanism.
- ELGIN has been validated both formally and experimentally, on a variety of problems.

Contributions

- As a concept learner, ELGIN is
 - Formally justified under
 - * pac-learning (Valiant) framework
 - * mistake-bounded (Littlestone) framework [1]
 - Uses background knowledge
 - Capable of learning complex concepts using readily available background knowledge
- As an EBL system, ELGIN
 - solves the “multiple explanation” problem
 - contributes to use of approximations
 - contributes to use of multiple levels of generality
 - contributes to generalization-to- N [1]