Lazy Explanation-Based Learning: A Solution to the Intractable Theory Problem

Prasad Tadepalli*
(Prasad.Tadepalli@cs.cmu.edu)
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, and
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

Explanation-Based Learning (EBL) depends on the ability of a system to explain to itself, based on the domain theory, that a given training example is a member of the target concept. However, in many complex domains it is often intractable to do this. In this paper I introduce a learning technique called *Lazy Explanation-Based Learning* as a solution to the problem of intractable explanation process in EBL. This technique is based on the idea that when the domain theory is intractable, it is possible to learn by generalizing *incomplete* explanations and incrementally *refining* the over-general knowledge thus learned when met with unexpected plan failures. I describe a program that incrementally learns planning knowledge in game domains through Lazy Explanation-Based Learning. I present both empirical and theoretical evidence for the viability of Lazy Explanation-Based Learning.

1 Introduction

Explanation-Based Learning (EBL) systems learn by proving to themselves that a given training example is a member of a target concept. If they are successful in doing this, they generalize the instance to the class for which the same explanation holds [Mitchell et al., 1986, Dejong and Mooney, 1986].

However, in many combinatorially explosive domains like chess and circuit design, it may not be possible to prove that an example is a member of a target concept even if the system has a complete and correct domain theory. This problem is called the *Intractable Theory Problem* fMitchell et al., 1986, Tadepalli, 1986a]. Two-person games are an interesting domain where the explanation process is intractable because of the inherent uncertainty in the actions of the opponent. In particular, the complete explanation that a person has a forced win in two-person games involves exploring every possible action of the two players in the worst case. In fact, to learn only correct rules, even this much is not enough. Since many opponent's moves which are not applicable in the example may be applicable for problems in the generalized position, it is necessary to specialize the concept so that a given strategy works for all possible moves the opponent could ever make [Tadepalli, 1986b]. Assuming that the domain theory of the system simply consists of goals and primitive move definitions for each player, this involves exponential search even if the system is given a sequence of optimal moves played by the two players. In this paper, I propose a solution to the intractable theory problem and illustrate it using a working implementation.

The main idea in this paper is that when the domain theory is intractable, it is possible to learn by generalizing *incomplete* explanations, and to incrementally *refine* the over-general knowledge thus learned. In fact, most of the explanations that we give in everyday life are incomplete in the sense that they usually make a number of default assumptions [McCarthy, 1980]. E.g., consider the kidnapping story in [Dejong and Mooney, 1986]. The explanation of kidnapping is simply that John, the kidnapper, figured that one could make money by kidnapping a rich person's daughter and demanding ransom. This explanation is so incomplete that it does not even consider issues like the personal safety of the kidnapper. However, considering all possible hypothetical scenarios is computationally prohibitive. Our approach to this problem is to use the previously learned knowledge to reasonably constrain the hypothetical scenarios to a tractable minimum. Incomplete explanations, when generalized, lead to over-general rules. If the personal safety of the kidnapper is ignored in explaining the above example, the learned rule does not check for possible escape of the victim by threatening the kidnapper with a gun. When and if that happens, the system encounters an unexpected plan failure, which gives it an opportunity to refine its plans, provided, of course, it survives the surprise!

The next section describes our knowledge representation, and the latter introduces our learning technique called *Lazy Explanation-Based Learning* through a program that learns planning knowledge in two person games. The program is illustrated using examples from king and pawn endgames in chess. A complexity analysis of our learning algorithm and some empirical results on our program arc presented next. Section 6 describes some previous work related to Lazy Explanation-Based Learning, followed by a discussion of various tradeoffs involved in Lazy EBL systems. The paper concludes with a summary and directions to future work.

2 Knowledge Representation

A domain is described to the system in the form of a *domain theory* consisting of a set of goals for each player, a set of legal moves defined as STRIPS-operators, a set of horn-clauses which define the preconditions of operators in terms of more primitive predicates, and a set of *operational*

*This work is supported by the National Science Foundation under Contract Number IRI-8740522, and Defense Advanced Research Projects Agency under Contract Number N00014-85-K-0116. The opinions expressed in this paper are those of the author and do not reflect any policies either expressed or implied of any granting agency.

predicates of which the preconditions of the learned rules are to be composed.

Knowledge is represented by a set of inter-related goals and plans for each player. A goal consists of a definition expressed in a quantifier-free logic, a *sign* , and a number called *promise* that indicates the worth of the goal. Associated with each goal, there may be a set of *optimistic plans,* or *o-plans* that can be used to achieve the goal. The body of an o-plan is a sequence of generalized moves, each move being preceded by the weakest conditions that must be true of a state so that the rest of the move sequence is applicable in that state (cf. Table 2). Each move in the o-plan is associated with a player who makes that move. The order of moves in the o-plan is fixed, except that when two successive moves are to be made by the same player, it is assumed that there is an irrelevant move by the opponent between those two moves.[1] All the moves in the o-plan are *locally relevant* in that each move either directly achieves a part of the goal or enables another move that follows it by satisfying some of its preconditions.

O-plans are related to other o-plans through *sub-plan* and *counter-plan* relations. An o-plan *P* is a *sub-plan* of another o-plan *Q,* if, under some circumstances, *P enables* some conditions necessary for *Q.* Similarly, an o-plan *P* is a *counter-plan* of Q_t if, under some circumstances, achieving *P disables* some conditions necessary for *Q.*

To use an o-plan in planning, the free variables in the precondition of some suffix of the o-plan body must be instantiated. Because of the inherent uncertainty introduced by the moves of the opponent in two-person games, in many positions a single o-plan is not adequate to achieve one's goals. So the planner combines o-plans into more complicated *c-plans* using *plan combinators* such as *SEQ,* and *MESH.* Trivially, every instantiated o-plan is a c-plan. SEQ produces new c-plans by sequentially composing its component c-plans. MESH produces new c-plans by inter-'eaving its component c-plans in all possible ways. (See .Tadepalli, 1989] for details.) Similar plan combinators are used in [Bratko, 1984], and [Campbell, 1988]. The *complexity* of a c-plan is the total number of o-plans in the c-plan.

3 Lazy Explanation-Based Learning

In this section, I introduce a learning technique called *Lazy Explanation-Based Learning,* which is based on the idea that it is much easier to produce an *incomplete* explanation in many domains than it is to produce a *complete* explanation. Incomplete explanations, when generalized, give rise to *over-general* o-plans. In many cases, one might never need to revise the o-plans acquired in this fashion. However, in some cases, an o-plan might lead to an unexpected plan failure at which point the explanation is elaborated just enough to explain the failure, and a new counter-plan to the failed o-plan is learned.

Our method is embodied in a program called LEBL (Lazy Explanation-Based Learner), which is implemented in two person game domains. Our system has two main components: a *learner* and a *planner* (cf. Figure 1). In addition to the domain theory, the input to the learner consists of example games, i.e., a board position and a sequence of moves played from that position. Typically some goals of either player are achieved during the play. The output of the learner is a set of new o-plans and possibly new goals, and modifications to the old o-plans. The planner accepts a board position as input and outputs a (partial) solution tree for that position. The solution tree may be criticized by the teacher by playing with the system which helps the learner refine its o-plans. The learner calls the planner to explain the consequences of any alternative moves (i.e., moves other than those input by the user) of the two players generated using the previously learned o-plans.

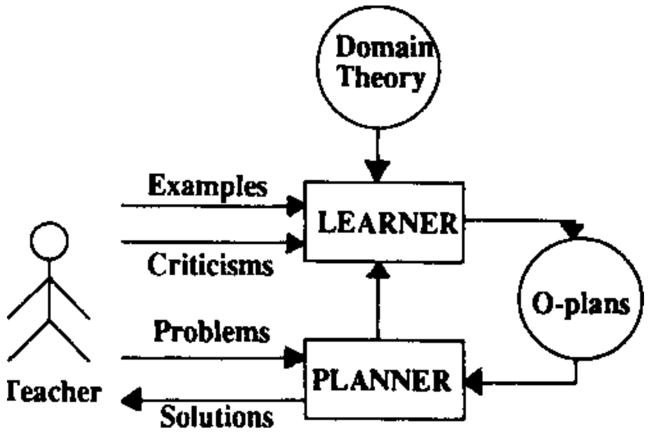


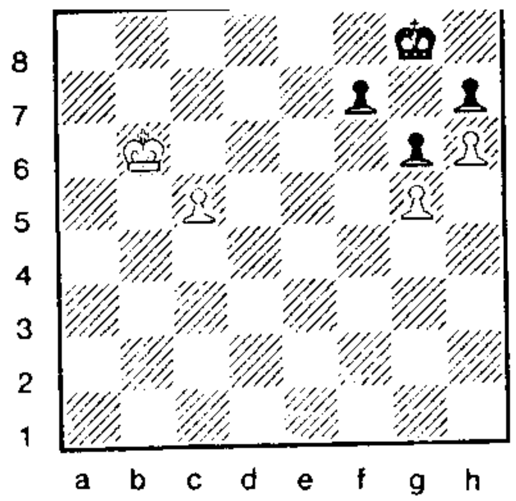
Figure 1: System Organization

3.1 Learning O-plans from Incomplete Explanations

A *complete explanation* that a board position is a forced win for a certain player, is a proof (or solution tree) that the min-max of that position evaluates to win. An *incomplete explanation* that a board position is a forced win for a player is a proof (partial solution tree) that the min-max of that position evaluates to win when the moves of the two players are restricted to some subset of all possible moves. An incomplete explanation is typically produced by considering only a few of all the possible moves at each intermediate state in the expansion of the state space. A crucial problem here is to decide which moves to consider at each node. Our system assumes that apart from the moves actually input to the system, the only moves relevant are those that occur in the c-plans generated by the planner using previously learned o-plans; i.e., the planner assumes that it knows *all* the o-plans necessary to explain all the relevant alternative moves by both the players. We call this assumption the *Omniscience Assumption.* This allows the system to limit its search only to what it reasonably expects with its current knowledge of o-plans, while considering more alternatives for both the players as it learns more o-plans.

The system is shown the chess position in Figure 2 and the subsequent line of play by the two players resulting in White's queening a pawn. Our top-level learning algorithm is described in Table 1. LEBL first checks that the move sequence is valid by proving that the preconditions of each move are satisfied when the move is made. The generalized versions of the proofs are also computed simultaneously in a manner described in [Kedar-Cabelli and McCarty, 1987]. Going backward from the final state, it then checks whether any goals in the system are satisfied for either player during the play. In our example, White's goal of queening his pawn was satisfied in the final state. If there is already an o-plan to queen a white pawn in the system and if this move can be

[1]As in [Fikes et al.f 1972], while planning, each o-plan is tried from the last suffix of its body toward the first until either the precondition of one of them matches the problem or none of the preconditions matches it. This has the advantage of trying a longer suffix of an o-plan only when no shorter suffix is applicable.



The play: 1. $c5 \rightarrow c6$, $g8 \rightarrow f8$ 2. $c6 \rightarrow c7$, $f8 \rightarrow e7$, 3. $c7 \rightarrow c8$

Figure 2: Example #1

interpreted as part of that, it makes that o-plan *active.* Since currently there is no such o-plan in the system, it creates a new o-plan with the postconditions initialized to the goal.

Each move *m* is tested to see if it enables the preconditions of any active o-plan, old or new. This is done by back-propagating the current preconditions of the active o-plan across the *m* and testing whether there is any change. If *m* enables the precondition of a new o-plan, as in the case of White's move in this example, it is added to the o-plan along with the generalized back-propagated precondition. If the move is an expected move of an old o-plan P, nothing is done. If the move is not expected in *P,* but still enables it, sub-plan links are created from each o-plan *P'* which contain that move to *P.* If no o-plan contains m, a new goal is created with the preconditions of *P,* and a new o-plan *P'* is started for this goal. Also, *P'* is made a sub-plan of *P.*

If a move has no effect on the preconditions of a new or old o-plan, that move is irrelevant to that o-plan, and is dropped from that o-plan. Since none of Black's moves in our example enables the preconditions of White's o-plan (except by yielding the turn to White), all moves of Black are considered irrelevant. All moves of White are relevant since they either directly achieve the goal (White's *c7* -> c8) or enable White's other relevant moves.

For each intermediate state in the move sequence, the learner calls the planner to search for any alternative plans that could have been used by either player leading to his goals. The planner works by alternately *generating* new c-plans for each player and *testing* them against all the old c-plans (of complexity < a user-given parameter *k)* of the opponent. The generation consists of combining its *current* library of o-plans (the Omniscience Assumption) using the plan combinators SEQ and MESH. The testing consists of expanding the game tree to include the moves consistent with the two plans being tested and reevaluating its min-max. The planner switches sides from a player if it is successful in finding a c-plan that changes the min-max value of the position in that player's favor. It terminates when it fails to generate any new c-plans of a given maximum complexity (3 in our experiments) that can improve the min-max value of the position. (See [Tadepalli, 1989] for more details on the planner.) After the planning is complete, the c-plans that occur in the solution tree of the two players are added to the corresponding active old plans.

```
Verify the input move sequence.
Let the active old plans AOP, and the
   new plans NP for the two players be ∅.
For each move m=(s,s') from the last to the first,
Do
   If m enabled a goal G for a player,
      If m is part of an old o-plan P for G, add it to AOP
      Else create a new o-plan, and add it to NP.
   For each P in {NP}∪AOP,
   Do
      new_prec(P) := Back-propagate (prec(P),m)
      If new_prec(P) is not the same as prec(P)
         If P is a new o-plan, add this move to P
         Else If m is a move not expected by P
            If m is part of a plan P' make P' a sub-plan of P
            Else If there is no sub-plan to P,
               if necessary, make new_prec(P) a new goal
               start a new o-plan P' for this goal, and
               make P' a new sub-plan for P
   End;
   Call the Planner to explore alternatives
   For each player, let AOP := AOP ∪
            {o-plans used in alternative paths}
End;
```

Table 1: Learning Algorithm

In this example, since the system does not have any o-plans in its Library at first, the planner returns with no additional moves explored. From this example, LEBL extracts a simple plan of pushing a pawn through from the 5'th rank of any column to the 8'th rank, and queen. After the back-propagation, the generalized preconditions for each move are *simplified.* The simplification consists mainly of removing the redundant preconditions, partial evaluation, and sorting the preconditions according to a predefined predicate order for efficient matching. Table 2 shows the o-plan *PL* learned from the first example.

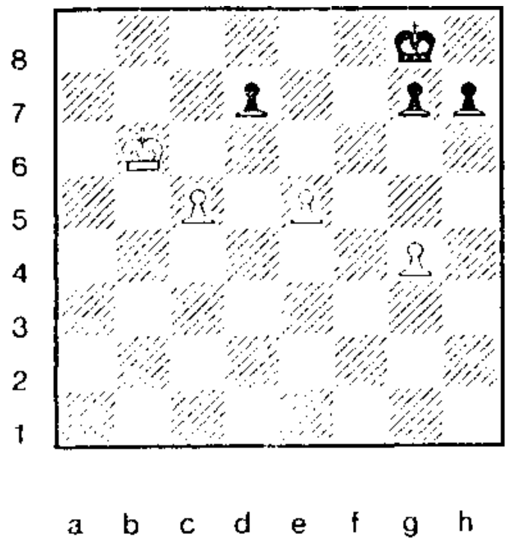
```
Player: WHITE
Goal:   QUEEN-WHITE-PAWN
Body:   [If ((ON ?p ?x 5) (TYPE ?p PAWN)
            (OWNS WHITE ?p) (FREE ?x 6)
            (FREE ?x 7) (FREE ?x 8))
         Then PAWN-MOVE(WHITE ?x 5 ?x 6)
         If ((ON ?p ?x 6) (TYPE ?p PAWN)
            (OWNS WHITE ?p) (FREE ?x 7) (FREE ?x 8))
         Then PAWN-MOVE(WHITE ?x 6 ?x 7)
         If ((ON ?p ?x 7) (TYPE ?p PAWN)
            (OWNS WHITE ?p) (FREE ?x 8))
         Then PAWN-MOVE(WHITE ?x 7 ?x 8)]
```

Table 2: O-plan PL1: to queen a pawn

The body of the o-plan *PL* consists of three rules, each corresponding to the position of the pawn in each of the intermediate states in the example including the first state. The left hand side of each rule describes the weakest conditions under which the rest of the o-plan can be executed provided no other o-plan of the opponent interferes with it. E.g., the first rule recommends pushing a white pawn in the fifth rank if the squares in the sixth, seventh and eighth ranks in the same file arc free. Similarly, the second rule says that a white pawn in the sixth rank must be pushed if the corresponding seventh and eighth rank squares are free, and so on.

3.2 Refining Over-general Knowledge

From our point of view, the most important thing to notice in the first example is that it is not fully explained how White could have won for all possible lines of play by Black. As a result, the plan *PL1* is over-general, and sometimes leads to unexpected failures as in the following example (Figure 3). Failures present the system with opportunities to refine its over-general plans. The refinement occurs by learning a new o-plan for the opponent and storing it as a counter-plan to the original o-plan.



White to Play

Figure 3: Example #2

In this example, we input the above chess position and ask LEBL to plan for White for at least a min-max value of 200.[2] LEBL outputs the following c-plan:

$\{(e5 \rightarrow e6\ e6 \rightarrow e7\ e7 \rightarrow e8)\}$ value: 200

The system can then be asked to play with the user, and it plays the first move in its plan, i.e., $<e5 \rightarrow e6>$. Then the user plays the move $<d7 \rightarrow e6>$, taking the pawn the system intends to queen. At that point LEBL notices that it cannot pursue its c-plan any further. It then,

1. Assigns blame to the first move that violated the preconditions of the remaining part of the unsuccessful o-plan (Black's move $<d7 \rightarrow e6>$ in this example).
2. Creates a new negative goal (by negating the violated preconditions of the unsuccessful o-plan) if there isn't already one that explains the failure of the first player's o-plan. In our example, there is already a negative goal of taking the opponent's pawn in the system, and hence no new goal is created.
3. Back-propagates the negative goal across the input move sequence and learns a new o-plan for the opponent. In our example, the body of this new o-plan *PL2* (see Table 3) consists of two moves: the first move is by White and it consists of pushing a pawn when there is a black pawn two ranks ahead in its left adjacent file. The second move consists of the black pawn taking the white pawn (in its right diagonal position).
4. Indexes the negative plan under the negative goal, and stores it as a counter-plan to the failed o-plan. (i.e., the new o-plan of Black, *PL2,* is stored as one of the counter-plans to *PL\.)*

```
Player: BLACK
Goal:   TAKE-WHITE-PAWN
Body:   If ((ON ?p2 ?X1 ?Y1) (ON ?p1 ?X ?Y)
           (TYPE ?p2 PAWN) (TYPE ?p1 PAWN)
           (OWNS BLACK ?p2) (OWNS WHITE ?p1)
           (PLUS1 ?X1 ?X) (PLUS1 ?Y2 ?Y1)
           (PLUS1 ?Y ?Y2) (FREE ?X ?Y2))
         Then (PAWN-MOVE WHITE ?X ?Y ?X ?Y2)
        If ((ON ?p2 ?X1 ?Y1) (ON ?p1 ?X ?Y2)
           (TYPE ?p2 PAWN) (TYPE ?p1 PAWN)
           (OWNS BLACK ?p2) (OWNS WHITE ?p1)
           (PLUS1 ?X1 ?X) (PLUS1 ?Y2 ?Y1)) :
         Then (PAWN-TAKE BLACK ?X1 ?Y1 ?X ?Y2)
Counter-plan-of: [PL1]
```

Table 3: O-plan PL2: To take White's pawn

From now on, the planner considers this o-plan of Black to refute White's o-plan to queen whenever it is attempted. So it does not propose the plan (e5->e6 *e6->el el ->e8)* again to queen the pawn in this example.[3] For reasons of modularity, LEBL stores this o-plan separately and links it to the original o-plan of White with counter-plan links instead of directly modifying the applicability conditions of White's o-plan.

4 Analysis of Algorithms

In our analysis, we assume that the c-plan complexity is bounded by *k.* We let *n* be the *average o-plan branching factor,* i.e., the average number of o-plan matches in each state, and let / be the average o-plan length. We make the assumption that the cost of proving an operator precondition is of the same order of complexity as proving an o-plan precondition. Let *p* denote the number of o-plans in the knowledge base, *g* denote the number of goals, and *s* denote the length of the input move sequence.

Our learning algorithm can be decomposed into two

[2]That means output any c-plan whose solution tree has a min-max value >200.

[3]Interestingly, it proposes the plan *(c5—>c6* c6 —»c7 *cl* —>c8) instead! The problem is that the notion of symmetry is not captured in the way the rules are represented in the system, and thus the two pawn captures by the black pawn have syntactically different explanations. After a subsequent, similar learning experience, the system learns to prepare for this kind of pawn capture as well.

parts: one part is checking that the move sequence is correct and extracting any goals and/or plans present in the sequence, and the second part is exploring alternative moves by calling the planner. The complexity of the first part of the algorithm is given by

$$O(l^2 \cdot max(p,g) \cdot n)$$

The planning cost at each intermediate state is (See [Tadepalli, 1989] for details):

$$O((2nl)^{2k}k \cdot p \cdot l)$$

Multiplying the above by *s,* the length of the input move sequence, and adding it to the cost of back-propagation, and simplifying, we have

$$\text{Learning cost} := O(l^2 \cdot max(p,g) \cdot \{(2nl)^{2k}k+n\})$$

In order to get some idea of the savings obtained in our system compared to an EBL system that learns from complete explanations, we compare its worst-case complexity to that of an α-β algorithm. It should be remembered, however, that even an α-β algorithm cannot be directly used to learn correct rules in games since the moves which arc not applicable in the example might be applicable in the generalization of the example [Tadepalli, 1986b]. However, the complexity of producing correct rules is at least as high as that of an α-β algorithm assuming that the program has no access to any other control knowledge. If the average branching factor of the game tree is b,[4]we have

$$\text{Complexity of } \alpha\text{-}\beta := O(b^s)$$

Since we assumed that the match costs in both the algorithms are of the same order of complexity, the major savings in LEBL is going to come from the exponent 2K, as opposed to *s* in α-β. In order to be able to find the solution that LEBL's planner finds, the α-β program must at least search for a depth that LEBL searches, which is *2lk,* Hence we can say that all other things being equal, LEBL performs better than α-β if the average length of the o-plans / is high. Further, it helps if the total number of o-plans p, the average o-plan branching factor n, and the maximum allowed c-plan complexity *k* are low. The above comparison reveals that Lazy Explanation-Based Learning performs better than an EBL program based on α-β search if the natural distribution of problems in a domain is such that a *high proportion* of the problems are solvable by c-plans of *low* complexity built from a *small* library of *long* o-plans with a *small* branching factor. In the next section, this claim is supported empirically by comparing the number of nodes searched by our program to that of a program based on α-β search.

5 Empirical Results

In order to compare the performance of LEBL to that of a program based on α-β search, I implemented a separate program called ABE (Alpha-Beta Explainer), which accepts a board position, and a move sequence as input, builds a complete solution tree for that position using the α-β algorithm, and outputs the number of nodes searched. ABE's evaluation function is the sum of the promises of all the goals in its domain theory which are satisfied in a given position. It is given one of the longest paths in the solution tree of the input position as a training move sequence and uses it to order the moves in its search (so that the move in

#	ABE	LEBL$_1$	LEBL$_2$	LEBL$_3$
1	>50,000	7	66	359
2	3,824	36	308	394
3	238	3	3	3
4	>50,000	18	155	155
5	16,124	32	59	38
6	992	816	75	75
7	5,415	15	657	657
8	5,876	232	281	281
9	36	2	10	10
10	37	2	10	10
11	36	5	10	10
12	36	6	10	10
13	36	4	4	4
14	155	3	7	7
15	157	3	5	5
16	212	3	3	3
17	3,405	60	125	125
18	10,138	242	250	250
19	12,623	250	255	255
# of new o-plans learned =		25	5	0
# of new goals learned =		5	0	0

Table 4: Nodes Searched in Learning

the training sequence is tried first when applicable). ABE limits its search to a fixed depth given by the length of the input move-sequence and is allowed to return any solution tree whose min-max value is at least as good as the value of the final state in the training sequence. The only advantage of LEBL over ABE is its knowledge of o-plans.

Initially, LEBL is given three goals for each player: queening a pawn which is worth 200, (or -200 if it is a Black's pawn), taking the opponent's king of worth 10,000 (or -10,000) and taking the opponent's pawn of worth 20 (or -20). We then trained LEBL on a set of 19 examples from the king and pawn endings. We input the examples one after another and recorded the number of states visited by the two programs. LEBL learned a total of 25 o-plans and 5 new goals in this training session. After training, LEBL is once again given the same set of training examples to see how much more search is involved in explaining the same examples when more o-plans are present. This time, LEBL learned 5 new o-plans and no new goals. The new o-plans learned by LEBL this time were for the goals learned in the first training session. We ran LEBL a third time on the same set of training examples and LEBL learned no new o-plans or goals.

The number of nodes searched by ABE and in the three training sessions of LEBL on the 19 examples are tabulated in Table 4. Ignoring the cost of matching the o-plans with problems, the number of nodes searched is a reasonable estimate of the search effort involved in learning. It is clear from the table above that LEBL consistently searched one to two orders of magnitude less number of nodes than ABE did on most of the problems tested. Even when the search effort of LEBL increased with the number of learned o-plans, it remained significantly *less* than the number of nodes searched by ABE. Surprisingly, in a few cases (e.g., Problem #6), the search *decreased* with learning more o-plans. The reason for this is that when there is no o-plan present in the system to achieve a goal, it tries to combine a number of o-plans to achieve it. This causes additional search which is avoided by having an o-plan which directly achieves the goal.

While training, LEBL is tested on a set of 8 new problems it has not been trained on, and each time, it is found to search at least an order of magnitude fewer nodes

[4]In our empirical experiments, b=5, and *w=2*

than the a-p search program. The number of errors as measured by an incorrect first move on the 8 test problems decrease gradually from 8 to 2, as LEBL learns from the 19 training examples. These results support our claim that LEBL performs reasonably well even by learning from incomplete explanations. It searches much fewer nodes than a program based on a-p search, and its errors on novel problems decrease gradually with learning.

6 Related Work

Prior to this work, [Pitrat, 1976] and [Minton, 1984J used methods similar to EBL to learn in game domains. PitraT' s program used procedurally encoded domain-specific heuristics to "simplify" the chess position before generalizing. Minton's program avoided some complexity by learning only from those move sequences in which all the opponent's moves are "forced". Minton found that the preconditions of the learned rules are too complex to be efficiently evaluated during the problem-solving, and suggested that one must try to learn rules that recommended plausible good moves rather than provably optimal moves. The work presented in this paper can be seen as an approach in that direction.

There are other EBL systems that make various simplifications and approximations to make explanations more tractable [Ellman, 1988,Chien, 1987, Bennett, 1987|. Ellman's program automatically makes assumptions like "ignore the history of the game" that greatly simplify the explanation. Chien's program uses defeasible assumptions about the persistence of certain facts during action sequences to simplify the explanations and refines the learned rules when it is faced with plan failures. This approach appears similar to Lazy EBL in many respects, one difference being that Chien's program, unlike ours, is mainly intended for single agent planning domains. Bennett's system deals with the intractability in mathematical reasoning by making simplifying approximations to mathematical formulae. While our work is consistent with all these approaches in general, it exploits a specific kind of simplification, that of giving an *incomplete* explanation, the extent of completeness being determined by the current knowledge of the system instead of by explicitly represented assumptions or simplification rules.

In [Doyle, 1986], Doyle presents a program that is able to repair inconsistent domain theories by reasoning and learning at multiple levels of abstraction. However, at each level of abstraction, the explanation is complete. Unlike his program, LEBL is not given any abstract version of the domain theory. The component of our program that learns from negative examples is similar in spirit to the programs described in [Mostow and Bhatnagar, 1987], [Gupta, 1987] and [Rajamoney et al., 1985]. When learning from failures, it is necessary to constrain learning in order to avoid learning too many uninteresting failure plans. LEBL addresses this issue by learning from failures only when a previously learned o-plan fails to achieve its goal.

7 Tradeoffs in Learning

Any learning system like ours must make several fundamental tradeoffs. One of them is the tradeoff between the learning time and the problem-solving (planning) time. The early EBL systems treated learning as forming schemata that can be directly instantiated during the problem-solving [Dejong and Mooney, 1986]. However, in complex domains like chess, it is simply not possible to learn a schema for every possible tactic, since that would require exorbitantly many schemata and many examples to learn them, instead, the problem-solver must be smart enough to flexibly apply its knowledge to the problem at hand. In our system, one way this tradeoff appears is as the question of when are o-plans composed. Our system makes the choice of learning only o-plans and composing them as needed during planning, thus requiring less training at the cost of more planning.

Another tradeoff our system exploits is between the learning effort for a single example and the number of examples needed to achieve a given competence. The conventional EBL systems completely explain each example, and hence spend more effort on each example. Lazy EBL systems distribute the explanation effort over several examples, thus requiring many examples to converge to the complete explanation produced by an EBL system. However, Lazy EBL performs better than an EBL system if the problem distribution is such that a rule learned from incomplete explanation is often adequate to make the correct predictions.

A third tradeoff, also discussed in [Ellman, 1988], is between the tractability of planning and the accuracy of the results. As our system learns more o-plans, the search space of the planner increases, and planning becomes less tractable. However, as the search becomes more exhaustive, the number of errors in planning decrease which means that the accuracy of planning increases. Thus, the system can be described as traversing the accuracy vs. tractability tradeoff curve in the increasing direction of accuracy.

8 Conclusions and Future Work

In this paper, we showed that one way to solve the intractable theory problem is to generalize from incomplete explanations and to refine the overgeneral plans when faced with plan failures. We illustrated this technique with an implemented program in two-person games and presented some empirical and analytical results to bolster our claims. While the program is, in theory, general enough to accommodate any two-person games whose moves are describable as STRIPS operators, it has only been tested in a simple version of king and pawn endgames of chess. We believe that this method can be generalized to single agent domains by mapping the game trees to AND/OR goal trees.

There are several open problems in our approach to intractable theory problem. One of the main problems is the expensiveness of match - an instance of the *utility problem* discussed in [Minton, 1988]. We observed that much time is spent in LEBL in matching problems with o-plans, and this could become a major source of inefficiency after a number of o-plans are learned. Unfortunately, the match problem is NP-hard, which means that there are no general ways of making it faster in the worst case. In [Tambe, 1988], Tarn be and Rosenbloom address this problem by restricting the expressive power of learned rules. Another promising approach, pursued in [Flann, 1989], is to learn more abstract (and hence, easy to match) high-level control knowledge to guide search in the planning space.

The problem of generalizing number and the structure of the explanation appear rather acutely in our domain. E.g., from the first example, a smarter program should be able to learn to queen a pawn from *any* rank. This is called the *Generalization-to~N Problem.* It appears that all the solutions proposed to solve this problem (see [Prieditis, 1986], [Shavlik and Dejong, 1987], and [Cohen, 1988]) can be implemented in the Lazy EBL in a straight-forward way.

There are also several theoretical questions related to the

Lazy EBL. E.g., how does the number of examples needed by a Lazy EBL system relate to the number needed by a normal EBL program to achieve the same level of competence? Is there a sense in which a lazy explanation-based learner may be said to converge, and if so, how many examples does it need to converge? We raised similar questions about EBL in [Mahadevan et al., 1988] and provided one possible way to answer them. We are looking for more comprehensive theoretical models that can answer some of these questions for Lazy Explanation-Based Learning.

Acknowledgments

I thank my advisor Tom Mitchell for encouraging me to work on this problem and for supporting and guiding me throughout this work. Many helpful suggestions are made by Neeraj Bhatnagar, Oren Etzioni, Haym Hirsh, Sridhar Mahadevan, Tom Mitchell, Jack Moslow, Lou Steinberg, Milind Tambe, Ming Tan, Chris Tong, and the anonymous reviewers of this paper. Special thanks are due to Jeff Schlimmer whose detailed comments greatly improved the readability of the paper.

References

[Bennett, 1987] Bennett, S. Approximation in Mathematical Domains. In Proceedings IJCAI-10, Milan, Italy, 1987.

[Bratko, 1984] Bratko, I. Advice and Planning in Chess Endgames. In Artificial and Human Intelligence, 1984.

[Campbell, 1988] Campbell, M. Chunking as an Abstraction Mechanism. Technical Report CMU-CS-88-116, PhD thesis, School of Computer Science, Carnegie Mellon University, 1988.

[Chien, 1987] Chien, S., Simplifications in Temporal Persistence: An Approach to the Intractable Domain Theory Problem in Explanation-Based Learning. Technical Report UILU-ENG-87-2255, University of Illinois, 1987.

[Cohen, 1988] Cohen, W., Generalizing Number and Learning from Multiple Examples in Explanation Based Learning. In Proceedings AAAI-88, St. Paul, Minnesota, 1988.

[Dejong and Mooney, 1986] Dejong, G. and Mooney R., Explanation Based Learning: A Differentiating View. Machine Learning, 2,1986.

[Doyle, 1986] Doyle, R. J. Constructing and Refining Causal Explanations from an Inconsistent Domain Theory. In Proceedings AAAI-86, Philadelphia, PA, 1986.

[Ellman, 1988] Ellman, T., Approximate Theory Formation: An Explanation-Based Approach. In Proceedings AAAI-88, St. Paul, Minnesota, 1988.

[Fikes et al., 1972] Fikes, R. E., Hart, P. E. and Nilsson, N. J. Learning and Executing Generalized Robot Plans. Artificial Intelligence, 3(4):251-288, Winter 1972.

[Flann, 1989] Flann, N. S. Learning Appropriate Abstractions for Planning in Formation Problems. In Proceedings of the International Machine Learning Workshop, 1989.

[Gupta, 1987] Gupta, A., Explanation-based Failure Recovery. In Proceedings AAAI-87, Seattle, WA, 1987.

[Kedar-Cabelli and McCarty, 1987] Kedar-Cabelli, S., and McCarty, L. T. EBG as Resolution Theorem Proving. In Proceedings of International Workshop on Machine Learning, pages 251-254, Morgan Kaufmann Publishers, Los Altos, CA 94022, June, 1987.

[Mahadevan et al., 1988] Mahadevan, S., Natarajan, B., and Tadepalli, P. A Framework for Learning as Improving Problem-Solving Performance. In Proceedings of Spring Symposium Series: Explanation-Based Learning, 1988.

[McCarthy, 1980] McCarthy, J., Circumscription - A Form of Non-Monotonic Reasoning. Artificial Intelligence, 13,1980.

[Minton, 1984] Minton, S. Constraint-Based Generalization: Learning Game-Playing Plans From Single Examples. In Proceedings AAAI-84, pages 251-254, Austin, TX, August, 1984.

[Minton, 1988] Minton, S., Quantitative Results Concerning the Utility of Explanation-Based Learning. In Proceedings AAAI-88, St. Paul, Minnesota, 1988.

[Mitchell et al., 1986] Mitchell, T., Keller, R. and Kedar-Cabelli, S., Explanation Based Generalization: A Unifying View. Machine Learning, 1,1986.

[Mostow and Bhatnagar, 1987] Mostow, D. J. and Bhatnagar, N. Failsafe - A Floor Planner that Uses EBG to Learn from its Failures. In Proceedings IJCAI-10, Milan, Italy, August, 1987.

[Pitrat, 1976] Pitrat, J. A Program for Learning to Play Chess. In Pattern Recognition and Artificial Intelligence, 1976.

[Prieditis, 1986] Prieditis, A. E. Discovery of Algorithms from Weak Methods. In Proceedings of the International Meeting on Advances in Learning, Les Arcs, Switzerland, 1986.

[Rajamoney et al., 1985] Rajamoncy, S., Dejong, G. and Faltings, B. Towards a Model of Conceptual Knowledge Acquisition Through Directed Experimentation. In Proceedings 1JCA1-9, Los Angeles, CA, 1985.

[Shavlik and Dejong, 1987] Shavlik, J., and Dejong, J. BAGGER: An EBL System that Extends and Generalizes Explanations. In Proceedings AAAI-87, Seattle, WA, July, 1987.

[Tadepalli, 1986a] Tadepalli, P. Learning in Intractable Domains. In Machine Learning: A Guide to Current Research, Los Altos, CA, 1986.

[Tadepalli, 1986b] Tadepalli, P. Learning Approximate Plans in Games. Technical Report ML-TR-8, Rutgers University, 1986. thesis proposal.

[Tadepalli, 1989] Tadepalli, P. Knowledge Based Planning in Games. Technical Report CMU-TR-89-135, Carnegie Mellon University, 1989.

[Tambe, 1988] Tambe, M., Rosenbloom, P. Eliminating Expensive Chunks. Technical Report CMU-CS-88-189, Carnegie-Mellon University, 1988.