Learning Activation Rules for Derived Predicates from Plan Examples

Dongning Rao[1], Zhihua Jiang[2,1], Yunfei Jiang[1]

[1]Software Research Institute, School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, P.R. China
[2]Department of Computer Science, Jinan University, Guangzhou 510632, P.R. China
E-MAIL: rdn2006@163.com, jnujzh@163.com, issjyf@mail.sysu.edu.cn

Abstract

Derived predicates are a compact way to depict complex planning domains, and their truth value in the current state is inferred from that of basic predicates via domain rules. However, domain rules designed by human experts often can not explain plan examples well in an intuitive way, and moreover, they can not explain plan examples when they are incomplete. In this paper, we develop a novel algorithm called LAR (Learning Activation Rules) for automatically discovering rules for derived predicates from a set of observed plans. Our empirical studies show that the conversion of activation sets of a derived fact is closely related to direct effects of actions, and therefore LAR attempts to build "activation rules" by tracing some special basic facts and their special state points in training examples. Indeed, a learned rule could be an approximate activation set of a derived fact, if training examples are sufficient and correct. We experiment the approach in the PSR (Power Supply Restoration) domain and evaluate the effectiveness of LAR empirically.

Introduction

There are some domains where plain STRIPS/ADL can not express the effects efficiently. PSR (Power Supply Restoration) domain is such a domain (Bonet & Thiébaux, 2003). In this case, derived predicates are introduced, which add additional effects (e.g., power outage of far away connected-line from the cut-off line) after each execution of an action. Derived predicates are a compact way to depict complex planning domains, and their truth value in the current state is inferred from that of basic predicates via domain rules.

However, domain rules designed by human experts often can not explain plan examples well in an intuitive way, and sometimes they even can not be guaranteed to be a correct and complete domain theory because of incomplete information. Often, a mass of deduction steps are needed in order to tell whether a derived precondition of an action holds in the current state under recursive rules for derived predicates. And moreover, if domain rules are incomplete, one can hardly explain why an observed plan is valid. Therefore, it is very interesting and promising that one can learn intuitive rules for derived predicates from observed plans in practice.

In the past, various approaches have been explored to learn rules from training examples. A well-studied territory is learning Inductive Logic Programming (ILP), which has been a hot research topic in the mid-90s (Lavrac & Dzeroski, 1994). A unifying theory of ILP is being built up around lattice-based concepts such as refinement, least general generalization, inverse resolution and most specific corrections. Sequential covering algorithm (Clark & Niblett, 1989) is one of the most widely used methods for learning a disjunctive rule set to cover all positive examples. It uses a subroutine called Learn-One-Rule to learn a single rule to cover some positive examples each time. This process is repeated until all positive examples are covered. FOIL (Quinlan, 1990) is an extension on sequential covering algorithm and can learn first-order rules and simple recursive rules. Besides, decision-tree or genetic algorithms are used to learn classification rules in order to judge whether a new instance satisfies a target attribute.

In this paper, we take the first step towards automatically acquiring rules for derived predicates from plans examples. Indeed, it is a difficult task assuming no predefined model of rules is known. Besides, learned rules must satisfy semantics characteristics of derived predicates. Here, we present an algorithm to learn rules only in proposition logic. For learning first-order rules for derived predicates is more difficult, we leave it as a major future extension. Our empirical studies show that a derived fact holds if only at least one of its activation sets is valid and one conversion of valid activation sets is caused by direct effects of actions. A basic fact can be an activation factor for a derived fact. If an activation factor holds in one state and does not hold in the latter state, then the activation set containing it is not valid any more and a new activation set is generated to make the derived fact hold if it is true. Thus we say a conversion occurs. So we attempt to build the conditions of a rule by tracing such a basic fact in training examples. Actually, such a learned rule could be an approximate activation set of a derived fact, if training examples are correct and sufficient.

The rest of this paper is organized as follows. The next section defines the problem of learning rules for derived predicates from plan examples. As one of our main contributions, the LAR algorithm will be presented and some experimental results will follow. After that, related work will be introduced. Finally, concludes with a discussion of future work will be addressed in the last section.

Problem Statements

In PDDL2.2 (Edelkamp & Hoffmann, 2004), the truth value of a derived predicate is determined by a set of domain rules in the form of *if* $\Phi_{\bar{x}}$ *then* $P_{\bar{x}}$, where $P_{\bar{x}}$ is the derived predicate and $\Phi_{\bar{x}}$ is a first-order formula. A planning problem containing derived predicates is defined as a tuple <Σ, I, G, B, D, R>, where Σ = <S, A, X> is a planning domain, I is the initial state, G is the goal state, B is the set of basic predicates, D is the set of derived predicates, and R is the set of rules for derived predicates in D. Sometimes, we call such a planning problem as a *derived planning problem*, a rule for a derived predicate as a *derived rule*, and the set R as a *domain theory*.

In order to include all the instances of a derived predicate which hold in a state s, a map relation is defined on s:

$$D(s) := \bigcap \{s' \mid s \subseteq s', \forall (P(\overline{x}), \phi(\overline{x})) \in R : \forall \overline{c}, |\overline{c}| = |\overline{x}| : s' \models \phi(\overline{c}) \Rightarrow p(\overline{c}) \in s')\}$$

$\models$ is the logical entailment under the closed world assumption on s’. Therefore, executing an action a in a state s will lead to a *successor state* s’: $s' = D(((s \setminus \bigcup del_a) \cup \bigcup add_a) \setminus DP)$, DP is the instance set of derived predicates which hold in the state s. $s'' = (s \setminus \bigcup del_a) \cup \bigcup add_a$ is called as an *immediate result state*. A *basic fact* is obtained by substituting each variable in the basic predicate with a constant, and so does a *derived fact* with respect to the derived predicate. A derived fact only can appear as a precondition of actions, called a *derived precondition*, or a goal, called a *derived goal*.

The concept of activation sets was first introduced by (Gerevini *el at.*, 2005), and implemented in a planning system, called LPG-td. An activation set for a derived fact is defined as a set of basic facts which can deduce the derived fact in the current state under applications of derived rules. Activation sets are calculated in the rule-graph and joined into the rule-action graph to drive the solution extraction process. However, activation sets defined by (Gerevini *et al.*, 2005) are temporary or state-dependent, for they need to be recalculated once the current state changes. Often, these recalculations are very time-consuming and unworthy. Therefore, for the convenience of learning rules, we must redefine activation sets which are expected to deduce derived facts doubtless under a given rule set, regardless of the current state.

Definition 1. Given a planning problem <Σ, I, G, B, D, R>, let d be a derived fact. An *activation set* for d is a minimal set F of basic facts such that F $\models^{R}$ d. And any member of F is said to be an *activation factor* for d.

An activation set defined above is stable and state-independent, and also can be calculated from a rule graph, which is a directed and cyclic AND-OR graph built from a grounded rule set (Gerevini *et al.*, 2005). A derived fact may have multiple activation sets, and they are composed of a set called Σ^*. Our early work has presented an effective algorithm to calculate the set Σ^* for a derived fact in a rule graph (for more details see (Jiang *et al.*, 2006)).

Example 1. Consider the domain PSR-MIDDLE-STRIPS_DERIVEDPREDICATES-P01_DOMAIN.PDDL from IPC-4 (http://andorfer.cs.uni-dortmund.de/~edelkamp /ipc-4/). A segment of the rule graph is depicted in Figure 1. For the simplicity, all rule nodes are omitted. The set Σ^* for the derived fact “NOT-AFFECTED-CB1” is showed as follows:

$$\left\{\begin{array}{l}\{\text{foo NOT-CLOSED-CB1}\}\\ \{\text{foo NOT-CLOSED-SD2}\}\\ \{\text{foo NOT-CLOSED-SD3}\}\\ \{\text{foo NOT-CLOSED-SD8}\}\\ \{\text{foo NOT-CLOSED-SD7 NOT-CLOSED-SD9}\}\\ \{\text{foo NOT-CLOSED-SD7 NOT-CLOSED-SD11}\}\end{array}\right\}$$

There are six activation sets for NOT-AFFECTED-CB1, which holds in a state only if anyone of its activation sets holds in that state.

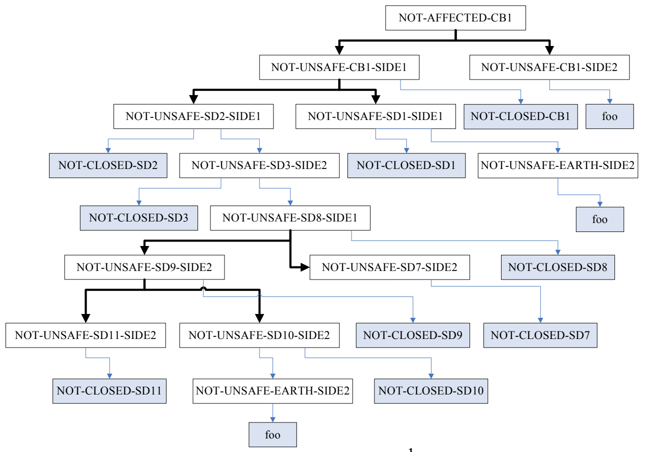

Figure 1: A segment of the rule graph[1] for P01_DOMAIN

In fact, an activation set for a derived fact corresponds to a rule for the derived fact. Formally, we define such a rule as follows:

Definition 2. Let F = {b_1, b_2…b_m} be an activation set for a derived fact d. Then we have an *activation rule* r_a: $b_1 \wedge b_2 \wedge \ldots \wedge b_m \rightarrow$d with respect to F. Similarly, let Σ^* be the set of activation sets for d, an *activation rule set* for d is

[1]The proposition in a white pane is a derived fact, and the proposition in a grey pane is a basic fact. Each directed edge starts from the conclusion of a derived rule and ends in the conditions of this rule. Moreover, two or more connected blank lines point to different conditions belonging to the same rule, respectively.

said to be: $R_a = \{r_a \mid r_a$ is the activation rule with respect to F, and $F \in \Sigma^*\}$.

Actually, an activation rule is not recursive; however, recursive rules designed by human experts often can not explain observed plans in practice well. The reasons come from two aspects: (1) a mass of deduction steps are needed for there are too many hidden derived facts, like NOT-UNSAFE-CB1-SIDE1. These hidden derived facts do not appear apparently in the initial state or any action model; (2) a domain theory may be incomplete, for instance, all rules about NOT-UNSAFE-CB1-SIDE1 are missing in Figure 1 and no activation set of NOT-AFFECTED-CB1 is known. Therefore, when NOT-AFFECTED-CB1 is a derived precondition or goal in a plan example, we will never know why the plan is valid. Therefore, it is significant to learn activation rules from plan examples.

Next, we formally define some performance measures for our learning problem. We consider a plan P as a sequence of actions, and a state s as a set of atoms.

Definition 3. A plan is said to be *correct* with respect to a rule set defining derived predicates, if each (basic or derived) precondition of an action holds in the state *D(s)* just before the action, and all goal propositions hold in the state *D(s)* after the last action.

In our learning problem, if a derived precondition of an action is not satisfied in the preceding state of the action in a plan example, then we say an *error* occurs, and we use E(a) = E(a) + 1 to count the number of errors with action a in P. Similarly, if a derived goal is not satisfied in the last state, then we say an error occurs, and we use E(G) to count errors in G. Then the Error_rate of P[2] is

$$Error\_rate(P) = \frac{\sum_{a\in P} E(a) + E(G)}{\sum_{a\in P} |derived\_precond(a)| + |derived\_goal(G)|}$$

Therefore, with respect to a set of plan examples Σ_P, the *Sample_error_rate* of a learned rule set R_a is defined as follows:

$$Sample\_error\_rate(R_a) = \frac{\sum Error\_rate(P)}{|\Sigma_P|}$$

On the other hand, a learned rule must be correct to be useful.

Definition 4. Given a derived planning problem <Σ, I, G, B, D, R> and let R be a complete domain theory, an activation rule $r = b_1 \wedge b_2 \wedge ... \wedge b_m \rightarrow d$ is *correct* with respect to R, if $\{b_1, b_2, ..., b_m\} \models^R d$.

Similarly, if a learned rule is not correct with respect to R, then we say an *error* occurs. Let R_a be a set of learned rules, and we use $E(R_a)$ to count errors with R_a. The *True_error rate* of R_a is:

$$True\_Error\_rate(R_a) = \frac{E(R_a)}{|R_a|}$$

[2]derived_precond(a) is the set of derived preconditions of action a, and derived_goal(G) is the set of derived goals in the goal state G.

The LAR Algorithm

In this section, we will discuss how to build an algorithm to learn rules for the learning problem described above from plan examples.

Extracting Training Examples

A plan example can not be a training example directly for our learning problem. However, if action models are totally known, then we can apply plan examples to the initial state, and obtain training examples from the result states.

Let us consider the domain in Example 1. Three solution plans are obtained from the planning competition 2004. We list these plan examples in Table 1.

	Plan1	Plan2	Plan3
s_0	$s_{1,0}$ = I	$s_{2,0}$ = I	$s_{3,0}$ = I
a_1	WAIT-2-0	WAIT-2-0	WAIT-2-0
s_1	$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
a_2	OPEN-SD7-0	OPEN-SD7-0	OPEN-SD9-0
s2	$s_{1,2}$	$s_{2,2}$	$s_{3,2}$
a_3	OPEN-SD11-0	OPEN-SD9-0	OPEN-SD7-0
s_3	$s_{1,3}$	$s_{2,3}$	$s_{3,3}$
a_4	CLOSE-SD3-0	CLOSE-SD3-0	CLOSE-SD3-0
s_4	$D(s_{1,4})$ = G	$s_{2,4}$	$s_{3,4}$
a_5		OPEN-SD8-0	OPEN-SD11-0
s_5		$s_{2,5}$	$s_{3,5}$
a_6		CLOSE-SD9-0	CLOSE-SD9-0
s_6		$s_{2,6}$	$D(s_{3,6})$ = G
a_7		OPEN-SD11-0	
s_7		$s_{2,7}$	
a_8		CLOSE-SD8-0	
s_8		$D(s_{2,8})$ = G	

Table 1: Three plan examples for P01_DOMAIN in PSR[3]

In Table 1, some propositions are basic facts, like "foo", "NOT-CLOSED-*" and "CLOSED-*", and other propositions are derived facts, like "FED-*" and "NOT-AFFECTED-*". $s_{i,j}$ means the immediate result state after action a_j in the ith plan. Apparently, the initial state I is also an immediate result state after the Meta-action START. Since there are only basic facts in the initial state and effects of actions, all immediate result states only contain basic facts. Actually, Table 1 shows the state-action sequence for a plan example. Next, we consider an example of action models in the table above:

```
(:action WAIT-2-0
:parameters ()
```

[3]**I**: (foo) (NOT-CLOSED-SD3) (NOT-CLOSED-SD6) (CLOSED-SD11) (CLOSED-SD10) (CLOSED-SD9) (CLOSED-SD8) (CLOSED-SD7) (CLOSED-SD5) (CLOSED-SD4) (CLOSED-SD2) (CLOSED-SD1) (CLOSED-CB2) (CLOSED-CB1). **G**: (FED-L11) (FED-L10) (FED-L8) (FED-L7) (FED-L6) (FED-L2) (FED-L1) (NOT-AFFECTED-EARTH) (NOT-AFFECTED-CB1) (NOT-AFFECTED-CB2) (NOT-AFFECTED-SD1) (NOT-AFFECTED-SD2) (NOT-AFFECTED-SD3) (NOT-AFFECTED-SD4) (NOT-AFFECTED-SD5) (NOT-AFFECTED-SD6) (NOT-AFFECTED-SD7) (NOT-AFFECTED-SD8) (NOT-AFFECTED-SD9) (NOT-AFFECTED-SD10) (NOT-AFFECTED-SD11)

```
:precondition
(and (AFFECTED-CB2) (NOT-AFFECTED-CB1))
:effect
(and (NOT-CLOSED-CB2) (not (CLOSED-CB2))))
```

In the definition of action WAIT-2-0, we can see that NOT-AFFECTED-CB1 is a derived precondition. Since WAIT-2-0 is applicable in the initial state I, NOT-AFFECTED-CB1 must hold in I. However, any member of the power set 2^I could be an activation set for NOT-AFFECTED-CB1. Therefore, we would better use the pair <I, NOT-AFFECTED-CB1> as a positive training example.

For the convenience of the algorithms, we formally define a training example as follows:

Definition 5. A *positive training example* is a pair < $s_{i,j}$, d>, where $s_{i,j}$ is the immediate result state after action a_j in the ith plan, and d is a derived fact which holds in $s_{i,j}$.

Next, we present an intuitive algorithm to generate training examples from a set of plan examples.

Algorithm 1 Generating training examples

```
Input: a planning problem <Σ, I, G, B, D, R>, and a set of plans {P_1, P_2…P_N}
Output: a set of training example T
Begin
1.  T ← ∅
2.  For i = 1 to N do
3.      $s_{i,0}$ ← I
4.      For j = 1 to | $P_i$| do
5.          For each derived precondition d of action $a_{i,j}$ do
6.              T ← T ∪ {< $s_{i,j-1}$, d>}
7.          $s_{i,j}$ is the immediate result state after applying action $a_{i,j}$ in the state $s_{i,j-1}$
8.          If j = | $P_i$| then
9.              For each derived goal g ∈ G do
10.                 T ← T ∪ {< $s_{i,j}$, g>}
End
```

Figure 2: Pseudo code for generating training examples

Learning Activation Rules

In a training example <s, d>, any subset of the state s could be an activation set for the derived fact d, but it does not make any sense for guessing one subset randomly to build an activation rule. The effects of an action affect the truth of basic facts, and thus the activation sets related to these basic facts are influenced indirectly, becoming valid or invalid. Consider the derived fact NOT-AFFECTED-CB1, whose activation sets have been found out in Example 1. In Table 2, we list all activation sets for this derived fact in each of its training examples from three plans in Table 1. Due to the limitations of space, training examples are shortened with the state name, since they are for the same derived fact. Besides, a state contains too many atoms, so we only list the atoms which are directly affected by effects of an action.

Training Examples	State (Atoms affected by an action)	Activation Sets [4]
$s_{1,0}$	I[5]	F_1
$s_{1,1}$	(NOT-CLOSED-CB2) (not (CLOSED-CB2))	F_1
$s_{1,2}$	(NOT-CLOSED-SD7) (not (CLOSED-SD7))	F_1
$s_{1,3}$	(NOT-CLOSED-SD11) (not (CLOSED-SD11))	F_1 , F_2
$s_{1,4}$	(CLOSED-SD3) (not (NOT-CLOSED-SD3))	F_2
$s_{2,0}$	I	F_1
$s_{2,1}$	(NOT-CLOSED-CB2) (not (CLOSED-CB2))	F_1
$s_{2,2}$	(NOT-CLOSED-SD7) (not (CLOSED-SD7))	F_1
$s_{2,3}$	(NOT-CLOSED-SD9) (not (CLOSED-SD9))	F_1 , F_3
$s_{2,4}$	(CLOSED-SD3) (not (NOT-CLOSED-SD3))	F_3
$s_{2,5}$	(NOT-CLOSED-SD8) (not (CLOSED-SD8))	F_3 , F_4
$s_{2,6}$	(CLOSED-SD9) (not (NOT-CLOSED-SD9))	F_4
$s_{2,7}$	(NOT-CLOSED-SD11) (not (CLOSED-SD11))	F_4 , F_2
$s_{2,8}$	(CLOSED-SD8) (not (NOT-CLOSED-SD8))	F_2
$s_{3,0}$	I	F_1
$s_{3,1}$	(NOT-CLOSED-CB2) (not (CLOSED-CB2))	F_1
$s_{3,2}$	(NOT-CLOSED-SD9) (not (CLOSED-SD9))	F_1
$s_{3,3}$	(NOT-CLOSED-SD7) (not (CLOSED-SD7))	F_1 , F_3
$s_{3,4}$	(CLOSED-SD3) (not (NOT-CLOSED-SD3))	F_3
$s_{3,5}$	(NOT-CLOSED-SD11) (not (CLOSED-SD11))	F_3 , F_2
$s_{3,6}$	(CLOSED-SD9) (not (NOT-CLOSED-SD9))	F_2

Table 2: activation sets for NOT-AFFECTED-CB1 in its training examples

NOT-AFFECTED-CB1 is a derived precondition of every action in plan examples and also a derived goal, so it is not surprising that its training examples include all result states in Table 1. Some interesting disciplines are observed from Table 2. First, in such a derived planning domain as PSR, actions are often used to generate an activation factor for a new activation set, or make an old activation set invalid. Second, one conversion of activation sets is caused by the change of the truth value of some basic fact. When the basic fact is made true, an activation set containing it is coming into being, and when it is made false, the activation set containing it becomes invalid. An example of such a basic fact is NOT-CLOSED-SD9. Third, some basic facts act as default activation factors, like "foo".

It is hard to explain how these disciplines work theoretically; however, our empirical studies show that they really work in the domain PSR or PROMELA. Next, we formally define some terms for the convenience of the presentation of our algorithm.

Definition 6. In the set of training examples for a derived fact, let $s_{i,j}$ and $s_{i,j+k}$ be the immediate result state after action $a_{i,j}$ and $a_{i,j+k}$ in the ith plan, respectively, and $k > 0$. A basic fact b is said to be a *conversion factor*, if b is a positive effect of $a_{i,j}$ and also a negative effect of $a_{i,j+k}$. And

[4] F_1 stands for {foo, NOT-CLOSED-SD3}. F_2 stands for {foo, NOT-CLOSED-SD7, NOT-CLOSED-SD11}. F_3 stands for {foo, NOT-CLOSED-SD7, NOT-CLOSED-SD9}. F_4 stands for {foo, NOT-CLOSED-SD8}.

[5] I is exactly same with I in Table 1.

we say that $s_{i,j}$ and $s_{i,j+k}$ are two special state points of b, $s_{i,j}$ is called as *T_point*, and $s_{i,j+k}$ is called as *F_point.*

An example of conversion factors is NOT-CLOSED-SD9, $s_{3,2}$ is its T_point, and $s_{3,6}$ is its F_point. Of course, a conversion factor may have more than one pair of special state points in a plan example. Conversion factors in the same training example set are often similar, like NOT-CLOSED-SD9 and NOT-CLOSED-SD8. Sometimes, we find that Definition 6 is too strict, so a relaxed case is considered. If a basic fact first holds in the initial state and then becomes a negative effect of some action in a latter state and is similar with other conversion factors, we say it is a *relaxed conversion factor*. An example is NOT-CLOSED-SD3 in Table 2, $s_{1,0}$ is its T_point, and $s_{1,4}$ is its F_point.

Definition 7. A basic fact is said to be an *invariable fact*, if it holds in the initial state but does not appear in any action model.

Next, we present the LAR algorithm to learn activation rules based on the disciplines above. The algorithm is feasible, no matter that whether the domain theory is complete. If a domain theory is complete, the True_error_rate of learned rules can be evaluated. If not, some important initial guidance is provided by learned rules for complementing the incomplete domain theory.

Algorithm 2 Learning activation rules (LAR)

Input: a planning problem <Σ, I, G, B, D, R>, and a set of training examples T

Output: a set of activation rules R_a

```
Begin
1.  $R_a \leftarrow \varnothing$
2.  Let $\{d_1, d_2 \ldots d_N\}$ be the set of derived facts contained in T, and T is grouped into $T_1, T_2 \ldots T_N$, with $T_i$ only containing the training examples corresponding to $d_i$
3.  For each rule $r = c_1 \wedge c_2 \wedge \ldots \wedge c_m \rightarrow d$ in R do
4.      If $c_1, c_2 \ldots c_m$ are basic facts and $d \in \{d_1, d_2 \ldots d_N\}$ then
5.          $R_a \leftarrow R_a \cup \{r\}$
6.  For h = 1 to N do
7.      For each (relaxed) conversion factor b in $T_h$ do
8.          $r_{general} = b \rightarrow d_h$
9.          $R_a \leftarrow R_a \cup \{ r_{general} \}$
10.         For each pair $< s_{i,j} = T\_point,\ s_{i,j+k} = F\_point >$ of b do
11.             Let $\{e_1, e_2 \ldots e_n\}$ be the positive effect set from action $a_{i,j}$ to action $a_{i,j+k-1}$ in the ith plan, which does not contain any conversion factor in negated form
12.             $r_{special} = e_1 \wedge e_2 \wedge \ldots \wedge e_n \rightarrow d_h$
13.             $R_a \leftarrow R_a \cup \{ r_{special} \}$
14. Remove all redundant rules in $R_a$
15. Find out the invariable fact set $F_{invar} = \{f_1, f_2 \ldots f_m\}$
16. For each rule $r = b_1 \wedge b_2 \wedge \ldots \wedge b_n \rightarrow d$ in $R_a$ do
17.     $r' = b_1 \wedge b_2 \wedge \ldots \wedge b_n \wedge f_1 \wedge f_2 \wedge \ldots \wedge b_m \rightarrow d$
18.     $R_a \leftarrow (R_a - \{r\}) \cup \{r'\}$
19. For each rule $r_{general}$ in $R_a$ do
20.     If there exists a rule $r_{special}$ in $R_a$ and $r_{special}$ is more special than $r_{general}$ then
21.         $R_a \leftarrow R_a - \{ r_{general} \}$
End
```

Figure 3: Pseudo code for learning activation sets

In the algorithm above, the training examples set T is built on state-action sequences from plan examples, so each grouping T_i ($1 \le i \le N$) can be sorted naturally by the state-action sequences. If the domain theory R contains some original activation rules, they should be joined into R_a to decrease the error_rate (step 3~5). The algorithm actually generates two kinds of activation rules, according to a conversion factor b and each pair of its state points: At T_point, b becomes true as a positive effect, which means that an activation set containing b is coming in being. Therefore, a most general rule $b \rightarrow d_h$ is built in step 8. At F_point, b becomes false as a negative effect, which means the activation set containing b is invalid, and another new activation set is functioning. So, a most special rule $e_1 \wedge e_2 \wedge \ldots \wedge e_n \rightarrow d_h$ is built in step 12, where $\{e_1, e_2 \ldots e_n\}$ is the set of all positive effects between the two state points but does not contain any conversion factor in negated form. Some reasonable adjustments for the learned rules are carried out in step 15~21. First, invariable facts are joined into each rule as default activation factors. Second, if there are two rules $r_1 = a \rightarrow c$ and $r_2 = a \wedge b \rightarrow c$, we can say r_1 is *more general* than r_2, whereas r_2 is *more special* than r_1. It is possible that some activation factors are omitted when we build a most general rule. Therefore, if there is a rule which is more special than a most general rule, the most general rule should be removed.

Considering an example, we try to learn activation rules for the derived fact NOT-AFFECTED-CB1, whose training examples have been listed in Table 2. According to the (relaxed) definition of conversion factors, there are three: NOT-CLOSED-SD9, NOT-CLOSED-SD8, NOT-CLOSED-SD3. Finally, we obtain the rule set as follows:

① foo ∧ NOT-CLOSED-SD3→ NOT-AFFECTED-CB1
② foo ∧ NOT-CLOSED-CB2 ∧ NOT-CLOSED-SD7 ∧ NOT-CLOSED-SD11→ NOT-AFFECTED-CB1
③ foo ∧ NOT-CLOSED-CB2 ∧ NOT-CLOSED-SD7 ∧ NOT-CLOSED-SD9→ NOT-AFFECTED-CB1
④ foo ∧ NOT-CLOSED-SD8→ NOT-AFFECTED-CB1
⑤ foo ∧ NOT-CLOSED-SD11→ NOT-AFFECTED-CB1
⑥foo ∧ NOT-CLOSED-SD7 ∧ NOT-CLOSED-SD11→ NOT-AFFECTED-CB1

Comparing with activation sets in Example 1, we can see that rule ①, ④ and ⑥ are totally correct, rule②, ③ and ⑤ are partially correct. A redundant condition NOT-CLOSED-CB2 appears in ② or ③, and a necessary condition NOT-CLOSED-SD7 is missing in ⑤.

Experiments

To validate the effectiveness of the algorithms, we develop a simple learning system, called DP^{LAR}. The input is the description of a planning problem (DOMAIN.PDDL and PROBLEM.PDDL) and a set of plan examples

(PLANS.TXT), and the output is a learned rule set (RULES.TXT). We use a planner called FF-DP which was developed in our early work (Jiang *et al.*, 2006), to calculate activation sets of a derived fact involved in a plan example. The results from FF-DP can be used to evaluate the true_error_rate of the learned rules.

Domain[6]	#t_examples	#rules	s_err_rate	t_err_rate
P01_Domain	294	126	7%	22%
P02_Domain	147	119	3%	25%
P03_Domain	290	196	3%	37%
P04_Domain	281	132	4%	31%
P05_Domain	284	106	2%	19%
P06_Domain	648	208	9%	30%
P07_Domain	307	156	6%	11%
P08_Domain	269	210	5%	26%
P09_Domain	303	108	3%	28%
P010_Domain	1197	315	2%	30%

Table 3: Experiments on PSR-Middle-StripsDerived

From the table above, we can see that the sample_error_rate of the learned rule set is acceptable. On the other hand, the true_error_rate is not satisfactory. The learned rules are not completely correct, even if most of them are very close to target rules. Therefore, some further refinement strategies should be considered to improve the precision of learned rules.

Related Work and Discussions

Many methods have been developed to learn rules in AI planning; however, to the best of our knowledge, few efforts have been put on learning rules for derived predicates. (Zettlemoyer *et al.*, 2005) has presented an algorithm to learn a model of the effects of actions in noisy stochastic worlds, while also learning derived predicates. They apply some predefined operators to literals to iteratively construct new predicates and test their usefulness in the learned action models. Indeed, they learn derived effects in an action model, not rules for derived predicates. Besides, Inductive Logic Programming (ILP) systems develop predicate descriptions from examples and background knowledge. The underlying techniques check every candidate literal and test its usefulness in covering positive examples. An essential assumption for this is that training examples are plentiful. However, in our experiment domains, the number of training examples for each derived fact is less than 50, and so these domains are not a well-suited application area for ILP systems.

Conclusions

In this paper, we have developed an algorithm for automatically learning rules for derived predicates from a set of plan examples even if the domain theory is incomplete. With conversion disciplines of activation sets of a derived fact in training examples, we can learn a set of "activation rules" that are nearly consistent with domain rules designed by human experts. We have shown empirically that it is feasible to learn these rules in a reasonable way using conversion factors and their special state points to construct most general or special rules. The learned activation rules can explain plan examples in a very intuitive way. While we take the first step towards automatically acquiring rules for derived predicates, there are a few limitations of the current work, which we plan to overcome in our future extensions. First, the learned rules are limited in proposition logic only for STRIPS plans examples, and it is an important task to extend them to be first-order rules for ADL plans. Second, to learn the domain from plans executed in the "real" world is very significative, which also implies learning probabilistic knowledge (as real world is noisy).

[6]#t_examples is the total number of training examples. #rule is the total number of learned rules. s_err_rate is the sample_error_rate of the learned rule set. t_err_rate is the true_error_rate of the learned rule set.

References

Lavrac, N., and Dzeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.

Clark, P., and Niblett, R. 1989. The CN2 induction algorithm. *Machine Learning*, 1989, 3, 261-284.

Quinlan, J.R. 1990. Learning logical definitions from relations. *Machine Learning*, 1990, 5, 239-266.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. *Technical Report 195*, Albert-Ludwigs-Universität, Freiburg, Germany, 2004.

Bonet, B., and Thiébaux, S. 2003. GPT meets PSR. *In Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, Trento, Italy, 2003, 102–111.

Gerevini, A., Saetti, A., Serina, I., and Toninelli, P. 2005. Fast planning in domains with derived Predicates: an approach based on rule-action graphs and local search. *In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, AAAI-Press, Pittsburgh, USA, 2005, 1157-1162.

Jiang, Z.H., and Jiang, Y.F. 2006. Planning with domain rules based on state-independent activation sets. *In Proceedings of Pacific Knowledge Acquisition Workshop of PRICAI06*, Guilin, China, 2006, 243-248.

Zettlemoyer, L., Pasula, H., and Kaelbling, L. 2005. Learning Planning Rules in Noisy Stochastic Worlds. *In Proceedings of Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains of IJCAI05*, Edinburgh, United Kingdom, 2005, 1-8.