Knowledge Acquisition for Planning with Incomplete Information *

Kang Soo Tae and Diane J. Cook
Department of Computer Science Engineering
The University of Texas at Arlington
Arlington, TX 76019, Box 19015
tae@cse.uta.edu, cook@cse.uta.edu

Abstract

As planning systems begin to solve more realistic problems, one of the most important and time-consuming tasks in developing a planning system is knowledge engineering (desJardins 1994). To autonomously learn planning knowledge, we propose novel approaches to solve some incompleteness problems arising in knowledge acquisition. We use an incremental learning method. We first introduce the *naive domain assumption* that every object of a certain type is functionally homogeneous. Using this assumption, an example is quickly generalized. This method drastically reduces the size of the search space required to acquire an operator definition. We conjecture that this assumption is currently adopted in most planning domains. The assumption, however, causes overly-general operator definitions in a realistic domain. To cope with this incompleteness problem, we make use of the ID3 machine learning system, to specify the incomplete definition by adding a literal that can classify non-homogeneous objects of a type into their own homogeneous subtypes. We show how to learn a unobservable predicate by this technique. A human tends to prefer positive predicates to negative literals. This tendency causes the problem of missing negative preconditions. We induce the negative literals by utilizing the relationship between the two-valued and the three-valued logic systems in planning. The learned negative literal helps prevent an inconsistent planning problem in a noisy state. The expert possesses knowledge not captured in a planning system (desJardins 1994). We developed a method to learn some implicit human knowledge such as opposite concepts by studying structural regularity among operators. This knowledge simplifies the operator definition. A multi-purpose intelligent robot should be able to adapt to a unknown task. When an operator is applied in a similar new domain and becomes overly-general, thus not fitting the domain, we introduce an operator splitting method. We first learn missing literals to specialize the operator to the new domain and if the operator is overly-specialized, we split the original operator into two operators.

This work is supported in part by National Science Foundation Grant IRI-9308308

Introduction

A planner is an inference engine acting like a shell of an expert system. To be commercially successful in the real world, a planning system should have provably rigorous operator definitions for an application domain to be integrated with a planner shell. However, human-generated operator definitions introduce many problems. First, knowledge engineering is expensive. Second, the expert's knowledge is incomplete in reality (desJardins 1994). Most planning systems are designed to operate with complete knowledge of all relevant aspects of the problem domain. Since all the facts in a world cannot be captured completely in a state, a state is an abstraction about a world ignoring some irrelevant details (Knoblock 1994). This abstraction tends to lead AI problems to toy problems when relevant complex details are ignored. Experts may overlook complex negative details in operator descriptions (Carbonell & Gil 1990). Third, the domain expert may not know exactly the actual capabilities of a robot. This requires verification of the robot's physical capabilities. By understanding its own action through experimentation, the robot can revise the initially incorrect or incomplete domain representation incrementally. This method can contribute to automatically learning realistic operator definitions for a complex domain. Fourth, implicit knowledge not encoded in a domain description causes some seemingly simple problems to become unsolvable. Tae and Cook (Tae & Cook 1995) solve some plan failures by automatically learning implicit rules from a human-generated representation.

To automate the acquisition of planning domain knowledge, we are currently building an experimentation-driven operator learning system, called WISER, which is running on top of PRODIGY. Given incorrect and incomplete background knowledge supplied by a domain expert, WISER learns operator preconditions using feedback from the environment. In this paper, we relax Wang's three assumptions that plague most current planning systems: 1) the obser-

vation is noiseless, 2) preconditions are mostly composed of predicates, and 3) everything in the state is observable (Wang 1994). We introduce a naive domain assumption to generalize an example in order to quickly learn operator definitions. These initial operator definitions are usually overly-general and cause incompleteness problems for real world applications. To cope with this problem, we introduce a machine learning method to refine the overly-general representation incrementally through experimentation in more complex environments. The positive and negative examples are detected by the learner's actual physical capabilities through experiments. ID3 is used to learn the missing literals that can separate positive examples from negative examples. The literal tends to be a unobservable concept. We propose novel methods to address some other incompleteness problems: learning missing operators in a similar but new domain given an incomplete set of operator definitions, and learning implicit human knowledge for inference using the structural regularity in operators. The knowledge can make the operator definition succinct.

Related work

Much of the existing research in planning assumes that domain operators are perfect. However, since in reality those expert-provided theories are incomplete, a fundamentally important area in planning is to autonomously learn and/or refine incomplete domain knowledge. Carbonell and Gil (Carbonell & Gil 1990; Gil 1992) explain a plan failure in terms of the inherent incomplete domain knowledge. EXPO refines the incomplete definitions by discovering missing literals through experiments when a planner's internal expectation and the external observation from the simulated environment differ. EXPO is applicable only for incomplete knowledge. OBSERVER (Wang 1995), running on PRODIGY (Carbonell *et al.* 1992), like EXPO, autonomously learns a new operator by observing an expert's problem solving method. The purpose of OBSERVER is to avoid the knowledge engineering problem. An initial incomplete operator is refined through the integrated cycles of planning, execution and plan repair. desJardins (desJardins 1994) developed two knowledge development tools: an operator editor that provides a graphical interface to build and modify incomplete and/or incorrect operators and an inductive learning system that acquires planning knowledge related with missing or incorrect preconditions via feedback from simulators or users.

EITHER (Ourston & Mooney 1990) and RAPTURE (Mahoney & Mooney 1993) are theory revision systems. EITHER is given an imperfect expert-supplied rule base and a set of correctly labeled examples. EITHER sends the positive examples that can not be proven and the negative examples that are proven to ID3, an inductive machine learning method using an information gain heuristic. EITHER revises its rule base as follows: Provably negative (false negative) examples are specialized by removing rules from the rule base and adding conjuncts to rules. Unprovably positive (false negative) examples are generalized by adding new rules, deleting conjuncts from rules, or adding new disjuncts. RAPTURE (Mahoney & Mooney 1993) combines connectionist and symbolic methods to revise the parameters and structure of a certainty-factor rule base. If the examples cannot be classified correctly, ID3's information gain heuristic is used to learn new terms and to change the structure of the rule base. Even though there has been a lot of research in the area of rule and concept learning and revision, there has not been much work related to extending the research to the area of planning. This paper is an attempt to integrate these two areas.

Naive Domain Assumption

Strips-like operator descriptions (Fikes & Nilsson 1972) model a robot's predefined actions in terms of a set of preconditions $pre(op)$, an add-list $add(op)$, and a delete-list $del(op)$. An operator includes a list of parameters, $op(v_1,..,v_m)$, for $m \geq 0$. Each parameter, v_i, is type-constrained. The domain theory, DT, specifies the legal actions in terms of a set of operators and inference rules. Inference rules represent deductions to be derived from the existing state. DT also specifies a type hierarchy for objects to reduce the search space. We will learn new inference rules for opposite concepts and new subtypes to represent unobservable literals.

Each type in DT's type hierarchy consists of a set of objects. Two objects of the same type map to the same variable: $var(ob1) \equiv var(ob2)$. Let two instantiated operators, op_i and op_j, be obtained from $op(v_1,..,v_n)$ by instantiating each parameter v_k of op to the same object respectively except that the r^{th} parameter is instantiated to different objects of the same type, say a for op_i and b for op_j. op_j is obtained by substituting a in op_i to b. These two instantiated operators are called r^{th} substituted-ops, represented as $subst(r, op_i/op_j, a/b)$.

Definition: Given $a, b \in ty$ and $\exists(op_i, op_j)[subst(r, op_i/op_j, a/b)]$ in a state S, let $S_i = \text{apply}(op_i, S)$ and $S_j = \text{apply}(op_j, S)$. a is functionally *homogeneous* to b with respect to op, $a \sim_{op} b$, *iff* $var(S_i) \equiv var(S_j)$ for all $S \in |S|$, the space of states. If a is homogeneous to b with respect to all operators, a is *homogeneous* to b,

$a \sim b$.

For two instantiated operators, if their instantiated preconditions are both satisfied (or both not satisfied) in a state and their respective actions have the same (variablized) effects on S, such that $var(add(op_i)) \equiv var(add(op_j))$ and $var(del(op_i)) \equiv var(del(op_j))$, we say that the two resources (or objects) a and b are homogeneous.

Theorem: A homogeneous relation $\sim$ on the set of objects in a type has the equivalence relation.

The equivalence relation for homogeneity $\sim$ measures the equality of objects in a type ty with respect to their applicability to a set of operators. If $\sim$ is satisfied on a subset of ty, defined by $[obj_e]=\{obj_i \in ty | obj_e \sim obj_i\}$, then $[obj_e]$ forms the subtype of the type. The n equivalence relations, $\{[obj_1], .., [obj_n]\}$, partition the set of objects of a type into n subtypes. We can choose an object randomly from a subtype $[obj_e]$, and use it as a typical representative for the subtype. If one resource of a subtype is tested for all the operators, no additional resource of the subtype needs to be tested further. Any object is equivalent to a variable.

Definition: A *naive domain* in learning is a domain where every object in a type belongs to one and the only one class.

For $a, b \in ty$, $subst(r, op_i/op_j, a/b)$ implies $a \sim b$ in the naive domain. This says that there is no advantage to selecting a over b in a naive domain. Therefore, we can experiment with an operator using only one object of a given type and variablize it without testing any more objects of the type. This assumption facilitates learning an operator definition from one example and reduces the search space drastically.

The generalization or abstraction of objects and concepts is inevitable in a real domain (Knoblock 1994) and the main problem is just to determine the most effective level of generalization for a given domain. Interestingly, 'the most conservative constant-to-variable generalization method' employed both by Gil and by Wang is a specific case of adopting the naive domain assumption. In both of these methods, an object is first initialized to a constant. If more than one object of a type is observed, the constant is generalized into a variable. The observation of at least two objects justifies the generalization of infinitely many objects of the type into one class of homogeneous objects. For example, because there is only one robot in a domain, the Robot-type is represented as a constant, and since there are many boxes, the Box-type is variablized. However, this method is problematic in a multi-agent environment. Suppose ROBOT1 is out of order and we need to use ROBOT2 for that domain. Unfortunately, we can not use the existing operator definition any more. On the other hand, our method generalizes ROBOT1 to ROBOT-TYPE and ROBOT2 can be unified to the existing definition. Thus, our method is more general. The advantage of this assumption is that a robot can expedite learning by reducing the size of the search space. However, this advantage comes at a price. The assumption of a naive domain is incomplete for more realistic situations. We use an incremental refinement approach to handle this.

Learning new terms in incomplete domains

A robot can expedite learning in simple environments first and then the robot incrementally adjusts its initial action in complex environments, composed of heterogeneous objects, by employing an inductive learning method. Let's start with an intuitive example of how a new-born baby learns to feed itself. Suppose the Feeding-World is composed of only two resources of two subtypes, b of a bottle-type and f of a finger-type, and the baby is endowed with feeding capabilities. He first starts with a naive world assumption where only one class [b], composed of the objects of bottle type, exists. He learns the preconditions {(in-mouth object) (hungry)}, and the the effect (not-hungry) after only one action is successfully applied. Soon, the baby may try f, its finger. The preconditions are satisfied, but the desired effect does not happen. Given that the baby is supplied with some reasoning ability (say ID3), he may begin to distinguish between the objects of the bottle-type [b] and the objects of the non-bottle-type [f] after a number of examples are given. He learns the new concept for [bottle-type] and the new preconditions, {(in-mouth object)(bottle-type object)(hungry)}.

Wang assumes that all the predicates in the state are observable. Wang's approach results in the following potential problem. A robot has a physical capabilities different from a human. A robot may carry an object up to 500 pounds in weight, while a human up to 100 pounds. Through observations, the robot will learn an overly-specific concept that it can carry the objects only up to 100 pounds. We will relax this assumption and divide the predicates into the observable (such as *height*) and the unobservable (such as *carriable*). For example, a robot cannot observe the weight of an object. Given a small number of objects, we can encode carriable(x) in our database. However, if there are infinitely many objects in the domain, it is impossible to

encode each object.

We introduce an autonomous method to learn these types of unobservable predicates incrementally. Each object is described as a vector of observable features, such as *(material shape color width length height)*. Each feature has the associated value. For example, the *shape* feature has a value from *(flat round angle)*. First, the robot learns overly-general operator definitions by assuming that every object is carriable in a naive domain. The robot experiences a plan failure due to the overly-general preconditions. Experimentation relies upon environmental feedback and produces a set of positive and negative (or false-positive) training examples for the preconditions of the faulty operator, such as *(+ (wood flat red narrow short high))* and *(- (metal flat red wide long high))*. We use ID3, an information-theoretic system, to classify a set of objects into homogeneous subclasses, say positive and negative examples, by selecting the features with the highest discriminating values in terms of the information gain. Information gain is the measurement of how well each feature separates a set of examples into their respective subclasses. (Cherkauer & Shavlik 1994; Quinlan 1986). The information needed to partition training examples is

$$I(n_1, ..., n_k) = -\sum_{i\ 1}^{k} \frac{n_i}{N} \log_2 \frac{n_i}{N}$$

where k is the number of subclasses, n_i is the number of examples in the subclass i, and $N = \sum_{i\ 1}^{k} n_i$. The expected information required for the tree with F as root is

$$E(F) = \sum_{j=1}^{w} \frac{\sum_{i=1}^{k} n_{ij}}{N} I(n_{1j}, ..., n_{kj})$$

where F has w possible values and n_{ij} represents the number of examples of subclass i having feature value j. Then the amount of gain by a feature F is

$$gain(F) = I(n_1, ..., n_k) - E(F)$$

The selected features are used for classifying unseen objects into respective subclasses. Our approach learns the functional concept composed of an unobservable predicate, such as *carriable*, by learning structural descriptions consisting of observable predicates. This structural concept satisfies the operational criterion of Mitchell et. al (Mitchell, Keller, & Kedar-Cabelli 1986) using only predicates which are easily observable by a robot. The benefit of learning an operational concept for a unobservable literal is that we don't need to rely on the information given by a human.

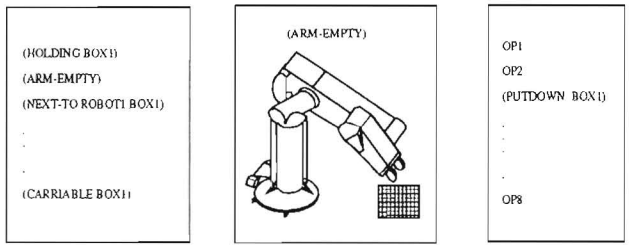

Figure 1: Noisy State and Plan Failure

Learning missing negative preconditions

Most current planners cannot handle noisy data. A noisy state causes a plan failure in Figure 1. The robot's arm is empty in the actual world, but the robot believes that it is holding *box1* and tries to execute a plan step, *(putdown box1)*. To make a planner robust in the noisy state, we propose a method to add negative constraints to a domain theory.

Knowledge acquisition is the mapping of human knowledge to a machine. Implicit knowledge is the knowledge which the expert believes the machine possesses after the mapping, but the machine actually does not possess. Implicit negative knowledge is difficult to detect and make explicit. For example, given that *(arm-empty)* is known to both a human and a machine, $\neg$*(holding x)* is known to the human, but not known to the machine. However, negative knowledge is crucial in handling complex real world problems. Carbonell and Gil (Carbonell & Gil 1990) show an incomplete domain theory in which a hidden negative constraint causes a plan failure. The failure is the inevitable result of an expert's poorly-designed domain theory. We propose a novel method to learn the necessary negative information at the initial design stage of a domain theory by unifying two types of logic systems as follows: A state description uses two-value logic: true or false. Our representation conventionally adopts the Closed-World Assumption. If p is not in the state, $\neg p$ is inferred. However, the operator description uses three-value logic: true, false, or irrelevant. If p is not in the preconditions, p is irrelevant. If p should not be in the state, $\neg p$ must appear in the preconditions. The knowledge acquisition of a learner is delimited by its language. The WISER description language is a typed first-order logic that allows conjunction and negation. Let PRED* represent all the predicates known to WISER. Let S be a positive state, P be the set of predicates that are true in S, and N the set of predicates which are not true in S. Since {PRED* - P} are not true by CWA, they correspond to N. Removing CWA, S transits to S*=P+Neg({PRED*−P}), where

Neg(X) means the negated value of X. S* is identical to S and provides a more comprehensive description of the same state. S* is used for inducing preconditions in this paper. A positive state is represented as the conjunction of predicates. This raw input data constitutes the potentially overly-specific preconditions for the operators in OBSERVER. It does not contain any negative information. Unlike Wang, we adopt three-value logic and transform the positive state to its closure to include the negative information before initializing it to overly-specific preconditions.

Let PRED*={A,B,C,D,E,F}, and the real preconditions Pre of an operator *op* be {A,B,C, ¬D}. In a positive state S_0={A,B,C,E}, OBSERVER initializes the preconditions as {A,B,C,E} and generalizes it to {A,B,C}. WISER initializes the preconditions to {A,B,C, ¬D,E,¬F} and generalizes it to {A,B,C, ¬D}. Given a state S_1={A,B,C,D}, Pre is not met, but *op* internally fires in Wang's method.

Let's give an example of noisy state problems. A human-generated domain theory is incomplete in handling these kinds of problems. The preconditions of *pickup* are {(arm-empty), (next-to robot box), (carriable box)} and the precondition of *put-down* is {(holding box)} in the Extended-Strips domain of the PRODIGY system. Suppose an inconsistent initial state is supplied by a noisy fast-moving camera: {(holding box1), (arm-empty), (next-to robot box1), (carriable box1)}. Surprisingly, if the goal state is {(¬ arm-empty)}, PRODIGY generates a plan, PICKUP, and if the goal state is {(¬ holding box)}, PRODIGY generates a plan, PUTDOWN. The result can be catastrophic in the real world. WISER successfully generates more constrained preconditions of PICKUP: {(¬holding box), (arm-empty), (next-to robot box), (carriable box)}. WISER uses Wang's algorithm (Wang 1995) to learn new effects. It checks if all the effects of the executed operator are true in the environment. We expand the algorithm by providing a method to refine incorrect effects in the later section.

Learning opposite concepts from operators

An operator corresponds to an action routine of a robot (Fikes & Nilsson 1972). Since each routine can be processed irrelevant of other routines, each operator is an independent unit in the domain theory. However, even though the operators are unrelated to each other on the surface, they are closely related in a deep structure of human percept. For example, the operators *open-dr* and *close-dr* are conceptually seen as opposites. Currently, no approach exists for examining this property between operators. In this section, we examine the structural relationships between operators and induce the opposite relationship between some operators and the opposite concept between some literals. The opposite concept will make an operator definition more succinct.

We represent the structural relationships as directed graphs, $D = (V, E)$, where $V = \{op_1, .., op_m\}$. An arc $e_{ij} \in E$ connects one operator op_i to another operator op_j if op_j can be always applied immediately after op_i is applied. Let *prestate*(*op*) be a state in which *op* can be successfully applied. After *op* is applied, *poststate*(*op*) is calculated as *prestate*(*op*) + *add*(*op*) − *del*(*op*). e_{ij} indicates that *poststate*(op_i) satisfies the preconditions of op_j. For example, given a set of operators, {*open-dr, close-dr, lock-dr, unlock-dr*}, there is an arc from *open-dr* to *close-dr*, but no arc from *close-dr* to *lock-dr* because a robot may need to subgoal to pickup a key.

We have two types of operators: destructive operators, such as *drill*, change the states of resources permanently, and non-destructive operators, such as *open-dr*, change the states temporarily. A change to a resource can be undone by another operator.

Theorem: A non-destructive operator belongs to a cycle.

Proof: Let *op* be a non-destructive operator. *poststate*(*op*) is *apply* (*op*, *prestate*(*op*)). If the preconditions of an operator still hold after the application of *op*, *prestate*(*op*) ⊆ *poststate*(*op*). Thus, there is an arc from *op* to itself as a vacuous self-loop. If *prestate*(*op*) ⊈ *poststate*(*op*), let $p = \{p_1, .., p_k\} \subseteq$ *prestate*(*op*) and $p \not\subseteq$ *poststate*(*op*). Since the resource R is not destroyed by definition, to execute *op* for R again, *prestate*(*op*) must be restored. Hence, there should exist a sequence of operators $op_j, ..., op_n$ that establishes p, where op_j immediately follows *op*. Thus, there is a path from *op* to op_n. Since the *poststate*(op_n) satisfies the *prestate*(*op*), there is a directed arc from op_n to *op*.□

For an n-cycle, a *poststate*(op_i) satisfies *prestate*($op_{(i+1) \bmod n}$) for $i = 1, .., n$. As a special case of an n-cycle, a 2-cycle is composed of two operators. They form a bipartite complete graph. They are called *dual* operators. Let op_i and op_j be dual operators, *Dual*(op_i) → op_j, and *Dual*(op_j) → op_i. One function of *Dual*(*op*) is to restore the preconditions of *op* by undoing the effects of *op*. Recursively, Dual(Dual(op)), which is op, restores the preconditions of Dual(op) by undoing the effects of Dual(op). Let op_i contain the add-list A and the delete-list D. Then, the add-list of op_j is D and the delete-list is

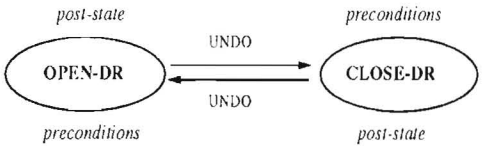

Figure 2: Graphical representation of two dual operators

A. $D - A$ is the formal definition of the undo function that restores $prestate(op)$ from $poststate(op)$ in a 2-cycle bipartite graph. Each dual operator op needs its $Dual(op)$, because $Dual(op)$ establishes the preconditions that op has destroyed. For example, *Open-dr* and *Close-dr* are dual because *open-dr* deletes its precondition $closed(dr)$, which is then achieved by *close-dr*. The duality of actions should be encoded in any planning domain. If op is a dual operator, its preconditions are not satisfied in $poststate(op)$. If $Dual(op)$ does not exist in the domain theory, the preconditions of op can't be restored. A rigorous proof is given in (Tae & Cook 1995). Consider a simple incomplete domain theory composed of one non-destructive operator, PICKUP that holds an object. Since there is no opposite operator that puts the object down, the robot will hold it forever.

Definition: If two operators, op_i and op_j, form a bipartite graph and undo each other, then op_i and op_j constitute opposite concepts.

PUTDOWN and PICKUP operators have the opposite functionalities of each other. Let op_i and op_j be opposite concepts of each other. $add(op_i)$ and $add(op_j)$ have the opposite functionalities of each other: are the opposites. If a literal $p \in A$ is the opposite concept to a literal $q \in A'$, such that $p \leftrightarrow \neg q$, a state $\{p, \neg p\}$ is feasible, but $\{p, q\}$ is not. To find $\{p, q\}$, such as $\{lock, unlock\}$ or $\{arm\text{-}empty, holding\}$, we use an experimentation method. We establish $\{p, \neg q\}$ as an initial state and change $\{p\}$ to $\{\neg p\}$. If such a change causes the change from $\{\neg q\}$ to $\{q\}$, p and q constitute the opposite concepts. $\neg q$ can be inferred from p. The opposite concept simplifies the operator definition. We can generate more simplified preconditions of PICKUP: {(arm-empty), (next-to robot box), (carriable box)}. This is intuitively more clear to humans.

desJardins (desJardins 1994) proposes as future research to develop automatic tools to verify an operator in relation to the rest of the knowledge base and to simplify operators syntactically by removing redundant preconditions. Our approach could partially contribute to that direction of research.

Learning new operators in a unknown domain

A general-purpose intelligent robot should be able to act in many domains including a unknown domain. Currently, a fixed domain theory is built separately for each domain. An operator successfully applied in one domain can ideally be reused in other domains for similar goals. If the operator becomes incomplete and/or incorrect in a new domain, the robot needs to refine it in the new domain and/or induce a new operator.

A training example is obtained by executing an operator in the environment (desJardins 1994; Gil 1992; Wang 1995). If the operator is successfully executed, it constitutes a positive example for the operator. A training example is composed of a sign, a state, and a goal. WISER is guided by positive and negative training examples in learning a new operator like a version space (Mitchell 1978). When an operator definition is overly-generalized and a negative (false positive) training example generates a plan, the example is used to specialize the definition. If the definition is overly-specialized. a positive (false negative) example, which can't generate a plan, is used to generalize it. WISER specializes an operator by adding a precondition or an effect like EXPO. WISER generalizes an operator by deleting a precondition like OBSERVER. However, if the specialization of an incomplete operator causes an over-specialization, EXPO or OBSERVER can't handle the complex problems such as an overly-specified effect. WISER splits the problematic operator into the overly-generalized and the overly-specialized operators: the type containing heterogeneous objects is split into homogeneous subtypes, each satisfying its own respective operator. If WISER cannot classify the objects in terms of the known type hierarchy, it induces new subtypes using ID3. This method is not covered in this paper, but it is similar to the one in Section 4. The splitting method and its example are presented.

Operator Splitting Method

1. Identify the over-general operator *gop*.
2. Specialize *gop* to *sop* by inserting a missing precondition or effect.
3. If a plan is generated for each positive training example, and no plan is generated for each negative training example, return *sop*.
4. *sop* is over-specialized. Find a subtype st_2 of the problematic type *ty* that correctly distinguishes the positive from the false negative examples for *sop*.

5. If no subtype can distinguish between positive and negative examples, send the examples to ID3 to find discriminating features and use them as a subtype, st_2.

6. Let st_1 be $\{ty \setminus st_2\}$. st_1 and st_2 partition *ty*. st_1 defines the resources that belongs to the positive examples for *gop* and st_2 does for *sop*.

7. Refine *ty* further to st_1 for *gop* and to st_2 for *sop*.

A robot is given the following operator definitions composed of a name, preconditions, and effects from the Extended-Strips domain.

- PICKUP: {(next-to robot ob) (carriable ob) (arm-empty)} {(del (clear ob) (arm-empty)) (add (holding ob))}
- PUTDOWN: {(holding ob)}, {(del (holding ob)) (add (arm-empty))}

The new task of the robot is to pick up a box and put it on another box. (It is the task in the Blocksworld domain.) This task is never done in the Extended-Strips domain, but it is similar to the known task of picking up a box and put it on the floor. A new domain usually introduces new (technical) terms not required in other domains. For example, (clear ob) and (on ob underob) are not necessary in the Blocksworld domain, but are necessary in the Blocksworld domain. New terms are likely to introduce some hidden problems. For simplicity, we assume here that WISER has diagnosed that *underob* cause some problem and that the type of *underob* consists of two subtypes: *table-type* and *box-type*.

Let the original Blocksworld domain theory consist of PICKUP, PUTDOWN, STACK, and UNSTACK operators. Note that we want to learn this theory from the Extended-Strips domain theory. A theory, perfect in one domain, becomes incomplete when applied in other domains. Note that in principle our approach is similar to EITHER's, which restores the original theory from a imperfect theory. PICKUP and UNSTACK are functionally similar: picking up an object OBJ1 located on another object OBJ2 if there is no other object on OBJ1. The only difference between them is the type of OBJ2: PICKUP is used if OBJ2 is the table-type. Otherwise, UNSTACK is used. (The same argumentation holds for PUTDOWN and STACK). However, their mechanisms seem to be the same: putting an object on top of another object. Thus, by the *naive domain assumption*, the robot initially hypothesizes that tables and boxes are homogeneous: OB1 can be put on the top of a box or a table. When this assumption does not hold, WISER experiments to understand why. Given a training example, T1:

- (+, ((next-to robot boxa) (carriable boxa) (arm-empty) (clear boxa) (clear boxb) (on boxb table) (on boxa table)), (holding boxa)),

WISER, running on PRODIGY, can generate and execute the one-step plan PICKUP(boxa). However, for a training example, T2:

- (-, ((next-to robot boxa) (carriable boxa) (arm-empty) (clear boxb) (on boxa table) (on boxb boxa)), (holding boxa)).

WISER can't execute the same plan PICKUP(boxa). WISER learns the new definition which includes the missing precondition (clear boxb) as briefly explained in section 5. [1] Let's call the new definition UNSTACK to conform to terms in Blocksworld domain. After UNSTACK is executed, the new effects (E1) is learned using Wang's algorithm (Wang 1995):

- ((del (on ob underob), (arm-empty)), (add (holding ob), (clear underob))).

Note that this newly learned effects are overly-specific since the effect, (clear underob), is not always true in the environment: For example, a planner generates the UNSTACK(boxa table) plan for (T1) as before, but with the different effects (E2):

- ((del (on ob underob), (arm-empty)), (add (holding ob))).

However, if underob is instantiated to a box, (clear box) is true and E2 is incomplete. If underob is instantiated to a table, (clear table) is not true. Thus, E1 is incorrect, EXPO can't handle this situation. To solve this dilemma, we need to reason by splitting the cases. E2 bound to a table is a positive concept for PICKUP, and E1 bound to a box is a positive concept for UNSTACK. By the *naive domain assumption* again, *underob* is bound only to the table-type in PICKUP, and to the box-type in UNSTACK. WISER splits the operator into two individual operators with the new constraints that *underob* is the table-type for PICKUP and the box-type for UNSTACK:

- PICK-UP:{(next-to robot ob)(carriable ob)(arm-empty) (on ob underob)(clear ob)}, {(del (on ob underob) (arm-empty)) (add (holding ob))}

[1] More discussion on this topic can be found in (Wang 1995).

- UNSTACK:{(next-to robot ob)(carriable ob)(arm-empty)(on ob underob)(clear ob)}, {(del (on ob underob) (arm-empty)) (add (holding ob)) (clear underob))}.

The learner now explores the operator PUTDOWN. Given a training example, T3:

- (+ ((clear boxc)(on boxb table)(holding boxa)(on boxc boxb)), (on boxa table)),

WISER can execute the plan PUTDOWN(boxa table). For a training example, T4:

- (- ((clear boxc)(on boxb table)(holding boxa)(on boxc boxb)), (on boxa boxb)),

WISER can't execute the generated plan PUTDOWN(boxa boxb) because of the missing precondition (clear boxb). The operator. is overly-generalized, and WISER adds (clear underob) to the preconditions. Let's call this specialized definition STACK. WISER can't generate a plan with STACK in T3 because (clear table) is not true in the environment. STACK is overly-specialized. WISER learns that (clear underob) with underob constrained to the box-type is a positive concept for STACK, and the literal with underob constrained to the table-type is a negative concept for PUTDOWN. The splitting operation produces PUTDOWN and STACK:

- PUTDOWN: {(holding ob)}, {(del (holding ob)) (add (arm-empty) (clear ob) (on ob underob))}
- STACK: {clear underob) (holding ob)}, {(del (holding ob) (clear underob)) (add (arm-empty) (clear ob) (on ob underob))}

Thus, WISER can restore the original domain theory from an imperfect domain theory like EITHER.

Conclusions and Future Research

As planning systems begin to solve more realistic problems, automated knowledge acquisition techniques are becoming increasingly important. We relax Wang's simplifying assumptions (Wang 1994) in order to face realistic problems. This relaxation introduces some incompleteness problems. Some state information is not observable to the system. We use ID3 to represent an unobservable concept by observable features. If the sensors of a mobile system are noisy, an erroneous state description occurs. In describing a state or a operator, people prefer positive predicates to negative literals and some expert's knowledge in the negative forms is not captured in the system. This introduces incompleteness to the system and the system may not perceive that a state is noisy. We extract some negative preconditions from a state by converting CWA to the three-valued logic. The negative constraints contribute in preventing anomaly situations in plan execution caused by the lack of negative human knowledge. We use the structural regularity to to learn implicit opposite concepts and use them for removing redundancy and making operator definitions simpler. Finally, we introduced an operator splitting method to learn missing operators in a new domain.

Empirical results demonstrating the planner's ability to generate successful plans before and after learning new knowledge occurs will be generated in the next step. We will further investigate the structural relationships between operators to infer some rules implicit in human mind.

References

Carbonell, J. G., and Gil, Y. 1990. Learning by experimentation: The operator refinement method. In *Machine Learning, An Artificial Intelligence Approach*, volume 3. San Mateo, CA: Morgan Kaufmann.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Knoblock, C.; Minton. S.; Perez, A.: and Wang, X. 1992. Prodigy4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Carnegie Mellon University, Pittsburgh, PA.

Cherkauer, K. J., and Shavlik, J. W. 1994. Selecting salient features for machine learning from large candidate pool s through parallel decision-tree construction. In Kitano, H., ed., *Massively Parallel Artificial Intelligence*. Menlo Park , CA: AAAI Press/The MIT Press.

desJardins, M. 1994. Knowledge development methods for planning systems. In *aaai-94 Fall Symposium Siries: Planning and Learning: On to Real Applications.*

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189 208.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation.* Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243 302.

Mahoney, J. J., and Mooney, R. J. 1993. Combining connectionist and symbolic learning to refine certainty-factor rule-bases. *Connection Science* 5:339–364.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47- 80.

Mitchell, T. M. 1978. *Version Space: An Approach to Concept Learning.* Ph.D. Dissertation, Stanford University.

Ourston, D., and Mooney, R. J. 1990. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence.*

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81- 106.

Tae, K. S., and Cook, D. J. 1995. Learning rules from incomplete and redundant domain theory. *Journal of Computing in Small Colleges* 10(5):242- 250.

Wang, X. 1994. Learning planning operators by observation and practice. In *Proceedings of The Second International Conference on Artificial Intelligence Planning Systems.*

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the 12th International Conference on Machine Learning.*