Learning New Planning Operators by Exploration and Experimentation

Yolanda Gil
Information Sciences Institute, USC
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu

Abstract

This paper addresses a computational approach to the automated acquisition of domain knowledge for planning systems via experimentation with the environment. Previous work showed how existing incomplete operators can be refined by adding missing preconditions and effects. Here we develop additional methods such as direct analogy to acquire new operators, decomposition of monolithic operators into meaningful sub-operators, and experimentation with partially-specified operators.

1 Introduction

In order for autonomous systems to interact with their environment in an intelligent way, they must be given the ability to adapt and learn incrementally and deliberately. Our approach is to improve initially hand-coded models for planning by failure-driven experimentation with the environment. Incompleteness is one possible fault in the given domain model. The initial domain may contain incompletely specified operators, and may be missing operators for legitimate actions that the planner can use to achieve goals. In previous work [1], we described the use of experimentation within the *Operator Refinement Method* (ORM) to find new preconditions and effects of existing operators. Shen and Simon [3, 4] describe methods to learn new operators: by exploring available actions whose effects are unknown, or by splitting an existing operator into two different ones when an expectation failure occurs. This paper presents some additional methods to acquire new operators. The first method presented is based on constructing new operators by direct analogy with existing ones through the types of the objects that they are applied to. Then we show how to create micro-operators, which contain only some of the preconditions and effects of a given operator. We describe two different methods to do this: building partial operators and sequencing. Our methods are goal-directed: they are triggered when the planner finds itself in a situation where it cannot solve a problem. The system assumes that the available knowledge is incomplete and tries the various methods to formulate new operators. Learning is always incremental, preferring overly incomplete specifications (that are progressively refined by the ORM) to more detailed specifications that may be incorrect. None of the methods is guaranteed to work, only the external execution of a proposed operator can prove its correctness. See [2] for more details on this work.

2 Direct Analogy

New operators can be learned by direct analogy with existing ones. As an example, suppose that the system has the knowledge about drilling holes shown in Figure 1(a). A hole can be made if a drill has a high-helix drill bit of the size of the desired hole and some cutting fluid, and if it is holding a part that has a spot hole in the appropriate location. Suppose now that the system is given the goal of producing a part with a hole in it, and there are no high-helix drill bits available. The preconditions of the operator for drilling cannot be achieved, and the planner is not able to solve the problem. But instead of returning a failure, our system uses the following reasoning to derive a new operator for drilling with other types of drill bits that might be available. The system finds that both high-helix and twist drill bits are of the same object type: DRILL-BIT, and thus it creates the new operator shown in plain font in Figure 1(b). The new operator only gets from the original one the types of the objects that it is applied to, and the effect that it is created for. Experiments are performed by executing the action under different conditions until a successful application is found. We describe in the next paragraph how the experiments can be designed efficiently. If the new operator cannot be applied successfully, then the process is repeated with other types of drill bits. If this does not yield any success either, then other object types are tried. In this case, a new operator for drilling holes with a milling machine is acquired when a different type of machine is considered. These experiments end when a successful application of a newly formulated operator is found that proves its existence. Once this happens, the ORM helps to locate additional conditions and effects that are specific to the new operator. They are shown with a star (*) in Figure 1(b). The method is summarized in Table 1. Notice that the power of this method comes from the possibility of relating P to P' through the object type hierarchy.

If a given problem cannot be solved by a set of operators because a precondition P that specifies the type of an object of an operator O cannot be achieved, formulate a new operator by direct analogy with O through P.

1. Find a related predicate. Look through the type hierarchy of the objects in the domain and find P' such that it refers to objects of the same type of the unachievable precondition P.
2. Formulate a new operator. Construct a new operator O' with the effects of O that the original problem subgoaled on and all the object types of O except P.
3. Experiment with the new operator. Execute the action. If the desired effects are not obtained, apply experimentation to isolate which of the other preconditions of O need to be added to O'. If O' is applied successfully in some state, then continue with step 4. Otherwise, go back to step 1, either looking for a different P' or considering a different P.
4. Refine the new operator. Apply the ORM to find all the preconditions and additional effects of the new operator.

Table 1: Method for learning a new operator by direct analogy with an existing one.

Choosing the right experiments is an important issue for making learning efficient. The conditions for the experiments are guided by the preconditions and effects of the original operator. If there are several operators for drilling that are available, then experiments that involve the preconditions and postconditions common to all drilling operations are preferred. The more available operators that already contain information about drilling, the more efficient the experiments designed to refine the new operator. Notice that these are heuristics and they do not make any guarantees about the convergence of the process.

```
(DRILL-WITH-HIGH-HELIX-DRILL
 (preconditions
   (and (is-a <machine> DRILL)
        (is-a <drill-bit> HIGH-HELIX-DRILL-BIT)
        (same <drill-bit-diameter> <hole-diameter>)
        (diameter-of-drill-bit <drill-bit> <drill-bit-diameter>)
        (has-fluid <machine> <fluid> <part>)
        (has-spot <part> <hole> <side> <loc-x> <loc-y>)
        (holding-tool <machine> <drill-bit>)
        (holding <machine> <holding-device> <part> <side>)))
   (effects (
        (del (is-clean <part>))
        (add (has-burrs <part>))
        (del (has-spot <part> <hole> <side> <loc-x> <loc-y>))
        (add (has-hole <part> <hole> <side> <hole-depth>
                       <hole-diameter> <loc-x> <loc-y>)))))
```

(a) An operator for drilling a hole using a high-helix drill bit

```
(DRILL-WITH-TWIST-DRILL
 (preconditions
   (and
        (is-a <machine> DRILL)
        (is-a <drill-bit> TWIST-DRILL-BIT)
      * (same <drill-bit-diameter> <hole-diameter>)
      * (diameter-of-drill-bit <drill-bit> <drill-bit-diameter>)
      * (has-spot <part> <hole> <side> <loc-x> <loc-y>)
      * (holding-tool <machine> <drill-bit>)
      * (holding <machine> <holding-device> <part> <side>)))
   (effects (
      * (del (is-clean <part>))
      * (add (has-burrs <part>))
      * (del (has-spot <part> <hole> <side> <loc-x> <loc-y>))
        (add (has-hole <part> <hole> <side> <hole-depth>
                       <hole-diameter> <loc-x> <loc-y>)))))
```

(b) New operator for drilling with a twist drill bit. The stars indicate new facts acquired by the Operator Refinement Method for the new operator.

Figure 1: Learning a new operator for drilling by analogy with an existing one.

3 Micro-operator Formation

New operators can also be acquired by learning useful partial specifications of an existing one. One possible way to do this is when the system encounters situations in which only some of the effects of the action are desired. If this is the case then experimentation is used to find if only some of the preconditions are required for the partial effects needed.

Suppose the system has the operator for cutting specified in Figure 2(a). The operator expresses that if a circular saw has a type of attachment called friction saw and some cutting fluid and if it is holding a part, then the size of the part can be reduced and the resulting surface is smooth. Now suppose that the system is given a problem whose goal is to make the size of a part smaller, and that no fluids are available in the initial state. The goal cannot be achieved with the available knowledge, and yet there is a way to solve the problem. The system formulates a new cutting operator that has only the effects that it needs from the original one, and only the preconditions that specify the type of the objects required for the operator. The action is then executed. If the desired effect is not obtained, then the system finds which additional conditions are required. This is done by experimenting with the action applying it under different situations. The experiments are guided by the preconditions of the known operator for cutting. This process ends when a successful application of the new operator is found (thereby proving its existence). This happens when the desired effect is obtained in a state where not all the preconditions of the original operator are true. Finally, the ORM is called to further refine the operator. The result is a cutting operator without the preconditions and effects that have to do with obtaining a reasonable surface condition quality (having fluid on the machine), as shown in Figure 2(b). This method for learning a *partial* operator is summarized in Table 2.

```
(CUT-WITH-CIRCULAR-FRICTION-SAW
 (params (<machine> <part> <attachment> <holding-device> <dim> <value>))
 (preconds (and
   (is-a <part> PART)
   (is-a <machine> CIRCULAR-SAW)
   (is-a <attachment> FRICTION-SAW)
   (has-fluid <machine> <fluid> <part>)
   (size-of <part> <dim> <value-old>)
   (smaller <value> <value-old>)
   (side-up-for-machining <dim> <side>)
   (holding-tool <machine> <attachment>)
   (holding <machine> <holding-device> <part> <side>)))
 (effects (
   (del (has-fluid <machine> <fluid> <part>))
   (add (surface-finish-side <part> <side> SMOOTH))
   (add (size-of <part> <dim> <value>)))))
```

(a) An operator for cutting

```
(CUT-TO-SIZE
 (params (<machine> <part> <attachment> <holding-device> <dim> <value>))
 (preconds (and
   (is-a <part> PART)
   (is-a <machine> CIRCULAR-SAW)
   (is-a <attachment> FRICTION-SAW)
*  (size-of <part> <dim> <value-old>)
*  (smaller <value> <value-old>)
*  (side-up-for-machining <dim> <side>)
*  (holding-tool <machine> <attachment>)
*  (holding <machine> <holding-device> <part> <side>)))
 (effects (
   (add (size-of <part> <dim> <value>)))))
```

(b) New operator for cutting to reduce the size. The stars indicate new facts acquired by the Operator Refinement Method for the New Operator.

Figure 2: Micro-operator formation when only some effects are needed.

A second possibility is *sequencing*, i.e. to detect a sequence of subactions that are currently represented by an operator. As an example, consider an operator to set up a machine for performing a machining operation. The operator would have several preconditions that check the availability of a machine, a holding device, a tool and a part. The set up consists of holding the tool in the tool holder, having a holding device on the machine, and holding the part with the holding device. Since a different setup is used for each machining operation, representing this set of actions as a single operator is an efficient way of expressing the configuration for the next operation. Now, suppose

When a given problem cannot be solved by the current operators because a precondition P of an operator O cannot be achieved, formulate a new operator O'.

1. Formulate a new operator. Construct a new operator O' with the desired effect and the type of the objects in O.
2. Experiment with the new operator. Execute the action. If the desired effects are not obtained, apply experimentation to isolate which of the other preconditions of O (not including P) need to be added to O'. End the process when O' is successful in a state where the preconditions of O are not true.
3. Refine the new operator. Use the ORM to find additional preconditions and effects of O'.

Table 2: Method for learning a new operator by micro-operator formation

that we want to perform some manual operation on a part. We ask the system to find a plan to hold it. With the available knowledge, holding a part is not possible because there are no tools that can be installed in the machine. But instead of returning a failure our system tries to find if the operator can be divided into a sequence of actions, one of them involving only holding the part. The operator to do the setup gives several independent operators: setup the holding device, hold the part, and setup the tool. Sequencing is done by following the same basic steps shown in Table 2, but in this case additional operators are formulated with the effects not originally needed.

4 Discussion

The methods presented in this paper have been implemented to demonstrate the feasibility of learning by experimentation. They are triggered when a lack of domain knowledge is detected, but the subsequent experimentation process is simulated manually. The full experimentation process (as is described in [2]) is implemented only for learning new preconditions and new effects. Work is underway to fully integrate the system. Other extensions include relaxing our assumption of deterministic environments where other agents cannot produce changes.

References

[1] Yolanda Gil. A domain-independent framework for effective experimentation in planning. In *Proceedings of the Eight International Workshop on Machine Leaning*, Evanston, IL, 1991. Morgan Kaufmann.

[2] Yolanda Gil. *Acquiring Domain Knowledge for Planning by Experimentation.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1992.

[3] Wei-Min Shen. *Learning from the Environment Based on Percepts and Actions.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[4] Wei-Min Shen and Herbert A. Simon. Detecting and correcting errors of omission after explanation-based learning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.