In Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96), May 29-31, 1996, Edinburgh, Scotland

Planning Experiments: Resolving Interactions between Two Planning Spaces

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

Abstract

Learning from experimentation allows a system to acquire planning domain knowledge by correcting its knowledge when an action execution fails. Experiments are designed and planned to bring the world to a state where a hypothesis (e.g., that an operator is missing a precondition) can be tested. When planning an experiment, the planner must take into account the interactions between the execution of the main plan and the execution of the experiment plans, since after the experiment it must continue to carry on its main task. In order for planners to work in such environments where they can be given several tasks, they must take into account the interactions between them. A usual assumption in current planning systems is that they are given a single task (or set of goals to achieve). However, a plan that may seem adequate for a task in isolation may make other tasks harder (or even impossible) to achieve. Different tasks may compete for resources, execute irreversible actions that make other tasks unachievable, or set the world in undesirable states. This paper discusses what these interactions are and presents how the problem was adressed in EXPO, an implemented system that acquires domain knowledge for planning through experimentation.

Introduction

Planning systems often make the assumption that omniscient world knowledge is available. Our approach makes the more realistic assumption that the initial knowledge about the actions is incomplete, and uses experimentation as a learning mechanism when the missing knowledge causes an execution failure. In a planning system, the inaccuracies of the knowledge base may render problems unsolvable or produce plans that yield unsuccessful executions. The imperfections of the domain knowledge have been closely related to planning and/or execution failures (Hammond 1986; Huffman *et al.* 1992), but they can also cause unexpected successful executions (Gil 1994). Planning systems that model a physical system and are given the ability to interact with it can directly examine the actual behavior of the physical system that the domain is supposed to model. This presents an opportunity for autonomous refinement of the imperfections of the domain model. Our approach combines selective and continuous monitoring of the environment to detect knowledge faults with directed manipulation through experiments that lead to the missing knowledge.

The first part of this paper summarizes our work on autonomous refinement of incomplete planning domains through experimentation (Gil 1994; 1993; 1992) and presents empirical results of its effectiveness and efficiency in improving the planner's domain knowledge when initial domain knowledge is up to 50% incomplete. Learning is selective and task-directed: it is triggered only when the missing knowledge is needed to achieve the task at hand. Our approach is based on continuous and selective interaction with the environment that leads to identifying the type of fault in the domain knowledge that causes any unexpected behavior of the environment, and resorts to experimentation when additional information is needed to correct the fault. The new knowledge learned by experimentation is incorporated into the domain and is immediately available to the planner. The planner in turn provides a performance element to measure any improvements in the knowledge base. This is a closed-loop integration of planning and learning by experimentation.

Research in the area of acquiring action models is mostly subsymbolic (Mahadevan and Connell 1992; Maes and Brooks 1990). An important component of our approach is the ability to design experiments to gather additional information that is not available to the learner and yet is needed to acquire the missing knowledge. Experimentation is vital for effective learning and is a very powerful tool to refine scientific theories (Cheng 1990; Rajamoney 1993), but other research on learning planning knowledge from the environment does not address the issue of experiment formulation and design (Shen 1993; Kedar *et al.* 1991). Previous work on learning by experimentation has not addressed the issue of how to choose good experiments, and much research on learning from failure has relied on background knowledge to build explanations that pinpoint directly the causes of failures. We want to investigate the potential of a system for efficient learning by experimentation without such background knowledge. Our approach uses domain-independent heuristics that compare possible hypotheses and choose the ones most likely to cause the failure. These heuristics extract information solely from the domain operators initially available for planning (incapable of producing such explanations) and the planner's experiences in interacting with the environment.

Planning experiments may interact with the main planning task that triggers learning. In the second part of the paper we describe how we handle these interactions: by giving the planner explicit guidance about how to conduct the experiment planning search. This is a general problem for planners that can be given two or more different tasks and must take into account the interactions between several planning search spaces.

Our approach to learning by experimentation has been implemented in a system called EXPO. EXPO's underlying planning architecture is the PRODIGY system (Carbonell *et al.* 1991) which provides a robust, expressive, and efficient planner. EXPO was tested with a complex process planning domain of dozens of operators, and a STRIPS-like robot planning domain that is widely used by planning researchers (both domains are described in (Gil 1992)).

Learning from the Environment by Experimentation

This section describes how EXPO detects that it is missing domain knowledge, then constructs a set of hypotheses of possible fixes, determines the more promising ones, and finally designs and executes experiments to determine the fix needed by its domain theory. More details can be found in (Gil 1994; 1993; 1992).

Detecting the Need to Learn

EXPO is given a suite of representative planning problems in the domain. This provides a purpose for learning: EXPO acquires knowledge that is needed to solve these kinds problems, instead of exploring the environment's behavior under various conditions. In a process planning domain, representative problems are to produce a part with a certain size, to give the surfaces of a part a certain degree of smoothness, to produce a part with a hole of specific diameter, depth, and position, etc. For each of these problems, EXPO creates a plan that solves the problem and starts executing the plan and collecting observations. EXPO monitors the external world *selectively* and *continuously*. Before the execution of an operator, EXPO expects the operator's known preconditions to be satisfied, so it checks them in the external world. If they are indeed satisfied, then EXPO executes the corresponding action. The operator's known effects are now expected to have occurred in the world, so EXPO checks them in the external world. For example, before executing grinding, EXPO would check the operator's known conditions, e.g., that the part is being held by a holding device, that there is a grinding tool installed in the grinder, and so on. After executing the action, EXPO checks that the part is of a smaller size and has a finished surface. Whenever its observations disagree with its expectations, EXPO concludes that it needs to learn in order to fix its knowledge about the operator so that observations and expectations coincide the next time the operator is used. This is the case in our grinding example if the surface is observed to be rough. After learning, EXPO's internal model should reflect more accurately the actual behavior of its environment.

Formulating Hypotheses

Next, EXPO hypothesizes a cause for the unexpected observation in terms of a fix to its knowledge base. Several possible categories of fixes to the domain knowledge can be considered, including the following:

- **Missing preconditions:** When an operator O is executed in state S and not all its known effects take place, EXPO considers the general hypothesis that the preconditions of O are incomplete and triggers learning to discover the missing condition C. Notice that C must have been true (by coincidence) every time that O was executed before. In our running example, suppose that the operator for grinding is missing a precondition to have cutting fluid in the machine (to absorb the heat produced by friction). Without fluid, the execution of grinding will produce a rough surface finish. When past executions of grinding succeeded, the machine had cutting fluid. But in those occasions the part had no holes in it, had a smooth surface, and was made of wood. Any of those could be a necessary condition for grinding to succeed. EXPO then engages in an experimentation process to discern which of those predicates is the missing condition.
- **Missing effects:** When a predicate P is found to have a value different than expected, EXPO considers the general hypothesis that some operator that was applied since the last time P was observed had the unknown effect of changing P. EXPO retrieves all operators executed since P was last observed, and considers them candidates for having incomplete effects. For example, if EXPO takes a part with a smooth surface and after grinding and drilling the part it notices that the surface is not smooth, then it will consider that either the grinding or the drilling operator is missing the effect of changing the surface quality. Experiments with these operators let EXPO observe P before and after each execution and determine which operator changes P.

Table 1 summarizes hypothesis generation for these two cases. Other possible fixes to the domain knowl-

what triggers learning	*general hypothesis*	*particular hypothesis*	*state to execute experiment*	*operator in experiment*	*observations in experiment*	
					before	*after*
unexpected outcome of O	O has an unknown precondition	The unknown condition is one of the predicates $\{P_i\}$ that were true in previous executions	Preconditions of O and P_i are satisfied	O	—	effects of O
unexpected value of P	P is an unknown effect of a previously executed operator	P is the unknown effect of one of the operators $\{O_i\}$ executed since last time P was observed	Preconditions of O_i are satisfied	O_i	P	P

Table 1: The design of experiments to learn new preconditions and new effects of operators.

edge include acquiring data about the state of the external world and acquiring new operators (Gil 1992; 1994).

Selecting Promising Hypotheses

Once the possible hypotheses are generated, EXPO needs to determine which one is responsible for the unexpected observation. The number of possible hypotheses may be quite large, and many may be ruled out by careful acquisition of informative instances through experimentation. In the process planning domain, the typical size of the hypothesis set is 50 to 100. In the grinding example, possible experiments are to try grinding a part that has a different number of holes, grinding a part that has a different surface finish, and grinding a part made of a different material.

Minimizing the number of experiments is important, not only because of the large numbers of hypotheses, but because there is a significant cost associated with each experiment. For each experiment the planner has to build a plan to set the environment in a state that satisfies many predicates. For example, in an experiment to try to grind a part with two holes the planner must select a part, make two holes in it, and then set it up in the grinding machine. Apart from the planning effort involved, the execution of those plans raises additional issues. Plan execution may use up valuable resources (including time), produce non-desirable changes in the environment that are hard (or impossible) to undo, and interfere with the goals of the system's main task. These issues are discussed briefly here to motivate the need to reduce the number of experiments, but they are central to the experiment search space and are discussed in more detail below.

The key to EXPO's efficient experimentation are a set of heuristics that help it concentrate on promising hypotheses. These heuristics, summarized in Table 2, are derived from the descriptions of other operators and of the operator in question. The locality heuristic points out, for example, that facts about the machine and tool used and the part being ground are more likely to be relevant to the failure. The presence of a steel part somewhere else in the machine shop is not likely to have affected the grinding operation. The similarity heuristic makes EXPO consider adding cutting fluid as a more plausible precondition of grinding than having holes in the part, because in this domain many operators that also reduce the size of a part require cutting fluid. Generalization of experience takes advantage of the fact that the conditions of the action must have been present in all past successful executions of it. For each operator, EXPO maintains a generalization of all the states where the execution was successful. The generalization is done using the operator's parameter bindings. Such generalization of the planner's past experience is useful to guide our search for the missing condition, because it contains the conditions that were common to all the states when the action was successfully executed before. The heuristics are described in more detail in (Gil 1992; 1993).

Designing and Executing Experiments

Each hypothesis is tested with an experiment. When the hypothesis is that the operator is missing a precondition P, an experiment is designed to test whether the operator will be successfully executed in a state where P and every precondition of O are satisfied, as shown in Table 1. This effectively becomes a goal for the planner, since a plan needs to be constructed to achieve the situation desired. This ***experiment search space*** is different from the main search space used to create the original plan that triggered learning. In searching for a plan, the planner must take into account the interactions with the experiment. This process is described in more detail below.

After the execution of this plan and of the experiment itself, observations are collected to determine what should be learned. If the hypothesis is confirmed, the domain knowledge is adjusted accordingly. Otherwise, the experimentation process is iterated until success or until no more hypotheses are left to be considered. Additional candidate hypotheses can be formed with the differences between S and a past state where O was successfully applied. If all those are also ruled out, the learner may still need to look for addi-

heuristic	*description*
locality of actions	objects affected by the action are likely to be already present in the operator's parameters
structural similarity	similar operators are likely to have similar preconditions
generalization of experience	necessary conditions have been present in all past successful executions of the action

Table 2: Domain-independent heuristics for suggesting better experiments.

tional candidates (for example, predicates that are not included in the state S because they were never observed), and even go back and consider an alternative general hypothesis, for example that O has conditional effects instead of a missing precondition.

After the operator is corrected, it will be used in any future planning. The main planning task can now be continued, and EXPO continues to watch for learning opportunities to correct its domain knowledge.

Empirical Results

EXPO was tested in two different domains: a robot planning domain frequently used in the planning literature, and a complex process planning domain with dozens of operators and states of large size. (Gil 1992) describes these domains in detail as well as other empirical results not shown here.

To control the amount of missing knowledge that EXPO was given in the tests, we first wrote a complete domain D with all the operators with all their corresponding conditions and effects. With this complete domain, we artificially produced domains D' with certain percentages of incompleteness (e.g., 20 percent of the preconditions are missing) by randomly removing preconditions or effects from D that EXPO could learn. Note that EXPO never has access to D, only to some incomplete domain D'.

Training problem sets and test problem sets were generated randomly. At certain points during learning, we ran the test set with learning turned off, and when EXPO made a wrong prediction the internal state was corrected to reflect the observations but no learning occurred (i.e., the domain operators were not changed.)

To show that EXPO is effective, i.e., that it can acquire new knowledge that is useful to the planner, we measured the cumulative number of wrong predictions (i.e., learning opportunities for EXPO). We also measured the number of problems in the test set that could be executed successfully to completion at several points during training. The following tables show results in the process planning domain, where the training sets had 100 problems and the test sets had 20 problems. Each problem required achieving several goals. The results are shown as cumulative round averages. In the process planning domain with 10 percent incompleteness the results were as follows:

training problems	*learning opportunities*	*test plans successfully executed*
0	0	5
10	0	16
30	8	17
50	10	19
100	10	19

and with 30 percent incompleteness:

training problems	*learning opportunities*	*test plans successfully executed*
0	0	1
10	17	8
30	23	13
50	29	18
100	30	19

EXPO always acquired a new precondition in every learning opportunity. Although EXPO did not acquire all the missing domain knowledge after learning, it learned in some trial runs the knowledge necessary to execute successfully the solutions to all the problems in the test set. In some cases, EXPO acquired knowledge that did not cause any improvement in the performance for the problems in the test sets. The number of test problems that were successfully completed always increased as learning progressed.

To measure the efficiency of EXPO, we measured the number of experiments that were needed using different strategies to select heuristics. We ran EXPO using each heuristic alone, using different combinations, and using no heuristics. When no heuristics were used, EXPO tried the candidate predicates in sequence.[1]

The results shown for this test are for the robot planning domain. The heuristics used are represented by a letter: g for generalization, s for structural similarity, and l for locality. Even though this domain is much smaller in size than the process planning domain, the

[1]We considered a divide-and-conquer strategy that recursively splits the candidate set, using $log(n)$ experiments to isolate the correct hypothesis (n being the number of hypotheses). The number of experiments needed with this strategy is comparable to (but still larger than) the number needed for our combined heuristics. However, for each experiment the planner must achieve many more additional goals because more conditions are tested in each experiment setup (Gil 1992; 1993).

number of possible hypotheses that EXPO could consider for each failure ranged between 50 and 85. With 20 percent of the preconditions missing, the cumulative number of experiments needed was as follows:

failures	*none*	*g*	*l*	*s*	*gls*
5	215	168	50	94	10
10	332	172	90	110	17

With 50 percent of the preconditions missing, the cumulative number of experiments needed was:

failures	*none*	*g*	*l*	*s*	*gls*
5	205	172	27	118	40
10	460	276	102	177	71
17	728	370	201	325	89

In all cases, the combination of the three heuristics reduced dramatically the number of experiments required, and yielded significantly better performance. Notice also that the number of experiments needed decreases as EXPO acquires more knowledge, because the heuristics are more effective when there is more knowledge about the domain available.

Planning and Executing Experiments

In order to perform an experiment, the world must be brought to a state where the conditions of the experiment are satisfied. For example, if the hypothesis is that the grind operator is missing the condition that the grinder has cutting fluid, we must reach a state where the current known preconditions of grinding and the hypothesized new condition are satisfied.

The planner must first come up with a plan to achieve this state from its current state, which is the state in which the failure that triggered experimentation occured. We call this process *pre-experiment planning.*

Once the pre-experiment plan is executed, the experiment can be carried out. In our example, we grind and check if this time the effects of the grind operator are obtained. If not, other hypotheses must be tested with other experiments. But if grinding works now, then the missing condition must be that the grinder has cutting fluid. The new condition is added to the operator GRIND. Then, the original plan that failed must be continued in order to achieve the original goal. If the pre-experiment plan has undone any of the facts necessary for the original plan, then a *post-experiment plan* is needed to restore those facts and continue with the main plan. Whether a post-experiment plan is used to enable the continuation of the original plan or replanning is done to achieve the original goals is not the issue here. The issue is that there is some effort needed to restore facts that were undone during pre-experiment planning.

Some pre-experiment plans are better than others depending on the criteria that are used. For example, minimal interference with the main plan may be an important concern. Suppose that learning was triggered when grinding part1 in the machine grinder1 with vise1 as a holding device and wheel1 as a tool. Then it would be better to use grinder2, wheel2, and vise2 with part2 in the experiments since vise1 is already holding part1. But perhaps we are more concerned with making the pre-experiment plan as short as possible, so we can learn as quickly as possible and go on with our main plan. If this is the case, using grinder1, wheel1, and vise1 would be better since they are already set up and ready for grinding operation. So, one experiment may be better than another one depending on what criteria are preferred.

EXPO designs experiments following a set of policies chosen by the user from a pre-defined pool. An example of a policy is to avoid using irreversible operators, since they can bring the world to a state where the main task cannot be achieved. Each policy in EXPO is implemented as a control rule in PRODIGY's language (Carbonell *et al.* 1991; Gil 1992). Control rules are used during the search at each decision point: to choose a node to expand, to choose a goal to achieve, to choose an operator to achieve a goal, and to choose bindings for the operator's parameters. Control rules can express prefereces among options, rejections of options, or select an option as the only possibility for a decision. The control rule that represents the policy to avoid irreversible operators is:

```
(RULE--REJECT-IRREVERSIBLE-OPERATORS
  (lhs (and (current-node <node>)
            (candidate-op <node> <operator>)
            (is-irreversible <operator>)))
  (rhs (reject operator <operator>)))
```

Each policy defines a preference to be used for decision making and can be thought of as a piece of control knowledge to be used during experimentation planning. Policies are grouped together to define overall strategies that the learner follows to plan experiments. We describe now EXPO's policies and strategies in detail.

Planning Experiments

All the policies that the user may define for the main planning task are also applicable to experiment planning. These policies correspond to the control knowledge (be it domain independent or not) given to the planner to be used for decision making in the domain. They can be considered *universal policies*, since they apply in both the main and the experiment search spaces. For example, we would consider an experiment that uses cheap materials to be better than another one that uses expensive materials. But the same principle applies in the main planning space. The quality of the experiment plans is determined in many dimensions by these universal policies that are to be addressed by research on how to measure plan quality, and are not discussed here. Experiment policies and universal policies may be in conflict. When this is the case, EXPO gives priority to universal policies unless indicated otherwise by the user.

The experiment policies defined for EXPO can be grouped under four topics: goal interactions, operator properties, binding interactions, and plan characteristics. The policies are cast in domain independent terms. They are summarized in Figure 1.

Goal Interactions—The goal interaction policies refer to the interactions between the goals in the experimentation space and goals in the main search space. They are different from the types of interactions within a search space. Here, a search path is preferred over another one it minimizes negative interference (or maximizes positive interference) with the top level goals. Notice that the preference is over which search paths to pursue, not over which goals.

Operator Properties—One policy is to avoid irreversible operators. Determining that an operator is irreversible requires proving that there is no plan that can undo its effects, which is undecidable. Also, the irreversibility of operators is not a binary feature: the same operator may be irreversible in some states and reversible in others. Because of these and other issues that make the automatic determination of irreversibility very complex, EXPO relies on a user-defined classification of operator's reversibility.

A second policy states that if the effects of operator O_1 are easier to undo than the effects of operator O_2, prefer O_1 over O_2. Determining the degree of reversibility of an operator is not a simple matter, so EXPO relies on an ordered list of operators defined by the user.

Another policy is to prefer operators that minimize state changes. If an operator O_1 has less effects than operator O_2, prefer O_1 over O_2. This policy is a more local version of another policy that prefers plans with fewer state changes (described below).

The next policy says that if an operator O_1 has a higher rate of success (based on the number of times that it has been used) than operator O_2, then prefer O_1 over O_2. The last policy avoids operators that have a rate of failure over a user-defined threshold. Notice that both policies try to use operators that have good models in the planner's knowledge base in order to avoid obtaining execution failures during the experiments.

Binding Interactions—During planning, the parameters of each operator are given values by binding them to objects in the current state. Some bindings may be preferable to others. For example, we may prefer to use in the experiments a different machine than the one that is being used in the main plan, since the machine used in the main plan is probably all set up for the operation. Other objects may not bring up such preferences. For example, if a brush is being used in the main plan to clean the metal burrs in the part we may not mind using it during the experiment planning. In summary, there may be different binding

- **Goal interactions**
  - **Avoid main goal protection violation**: If a search path clobbers a goal previously achieved by the main plan that is still needed to achieve the main goals, then prefer other search paths over this one.
  - **Avoid main prerequisite violation**: If a search path undoes a fact that the remaining main plan requires to be true, then prefer other search paths to this one.
  - **Support main goal concord**: If a search path achieves a goal that remains to be achieved by the main plan, prefer it over other paths.
- **Operator properties**
  - **Avoid irreversible operators**: Never use irreversible operators.
  - **Prefer easily reversible operators**: Prefer operators whose effects are easier to undo.
  - **Prefer operators that minimize state changes**: Prefer operators that have less effects.
  - **Prefer more reliable operators**: Prefer operators that have a higher rate of successful executions.
  - **Avoid unreliable operators**: If an operator's rate of failure is over a user-defined threshold, do not use it.
- **Binding interactions**
  - **Avoid objects of very high protection**: Never use objects that are used in the main plan and whose type is classified as very high protection.
  - **Prefer objects of lower degree of protection**: If two objects used in the main plan are being considered for binding the same variable, prefer the object with a lower degree of protection.
  - **Prefer least number of protected objects**: If several objects used in the main plan are being considered for binding different variables, prefer the set of objects that minimizes the total degree of protection.
- **Plan characteristics**
  - **Avoid long plans**: Never choose plans that are longer than a given length.
  - **Prefer short plans**: Prefer plans that are shorter.
  - **Avoid deep nodes**: Never expand nodes below a given depth. This maximum depth for the experimentation search must be given a value.
  - **Prefer shallow nodes**: Prefer expanding shallower nodes.
  - **Avoid plans with too many state changes**: Never choose plans that cause changes in the external world over a user-given number.
  - **Prefer plans with fewer state changes**: Prefer plans that cause a smaller amount of changes in the external world.

Figure 1: EXPO's experimentation policies.

preferences for different types of objects.

One interesting case in the process planning domain is the object part. Suppose that the main goal is to drill a hole of a certain width and depth in the part. Now suppose that the drilling operation fails because of a missing precondition, and experiments with the drilling operator are needed. If the experiments are done by drilling that part, we may not interfere with the main goal, but we would violate an implied goal: "Do not drill other holes in the part other than the ones specified in the goal". In fact, when we specify a goal to the planner in this domain (and many others) many such explicit goals are also desired but too complex to specify. A planner works by default on building a plan to achieve each of its given goals, so by default it would not interfere with these kinds of implicit goals. But since the experimentation process requires producing plans for other goals, such implicit goals may be violated by default. Notice that since the implicit goals are not part of the main goal description, they are not protected by the goal interaction policies. We have addressed this problem through policies for binding preferences as follows.

When a domain is defined, each type of object is assigned to one of the following classes:

- *Very high protection*: The instances of these types that are being used in the main plan are never to be used for the experiments.
- *High protection*: During experiment planning, other instances are preferred to instances of these types that are being used for the main plan.
- *Low protection*: During experimentation planning, other instances are preferred to instances of these types that are being used for the main plan, but instances of high or very high protection are never preferred.
- *Very low protection*: The instances of these types can be used any time during experiment planning.

In the robot planning domain there are only four types of objects, classified as follows:

- High protection: boxes
- Low protection: doors, keys
- Very low protection: rooms

The process planning domain is more complex, and has 33 types of objects, grouped as follows:

- Very high protection: parts
- High protection: holding devices
- Low protection: machines, machine tools, objects consumed during an operation.
- Very low protection: objects not consumed during an operation.

If necessary, the number of degrees of protection may be augmented, but the mechanism would be the same.

Once the protection classes have been defined, they are used to determine the policies that EXPO can use for choosing bindings: avoiding objects of very high protection, preferring objects of lower degree of protection, and preferring the least number of protected objects.

Plan characteristics—One criteria is to prefer shorter plans for the experiments. In PRODIGY, each level of a search involves the application of an operator or an inference rule. An inference rule represents a deduction from the current state, whereas an operator represents an externally executable action. The final plan is composed only of actions. This is why the depth of the search does not correspond to the length of the plan, and although they are related we may wish to control them separately. This is why four different policies are used to express preferences regarding the search depth and the plan length.

The last two policies prefer plans that minimize the number of state changes. The amount of changes that a plan produces in the sum of the effects of the operators that compose it. This policy is related to the policy that prefers operators that minimize state changes.

The Learner-at-Heart and the Planner-at-Heart

The experiment policies described in the previous section express different criteria that an experimenter may consider to design and choose experiments. Some of these policies may be conflicting, but the experimenter must have some overall, global strategy that determines which policies serve the strategy best.

With these policies, many different overall strategies may be designed. The following two strategies lie in opposite sides of the spectrum:

- **The Learner-at-Heart strategy**. This strategy has a more exploratory flavor and focuses on acquiring new knowledge. Novel sutuations are preferred over ones already experienced, and short experiment plans are preferred over longer ones that may delay learning.
- **The Planner-at-Heart strategy**. The main concern of this strategy is to accomplish the main planning task, acquiring new knowledge only if necessary to solve the problem at hand. Consequently, interactions with the main plan are avoided when possible, and using reliable operators is preferred over trying new ones.

Figure 2 summarizes the policies used to define each strategy.

Conclusion

Learning from the environment is a vital capability for an autonomous agent. The lack of knowledge affects the planner's capabilities, and learning requires both detecting a knowledge gap and determining a correction of the knowledge base. Experimentation is a powerful tool for gathering additional information from the environment that helps determine the appropriate correction. Our approach combines selective and continuous monitoring of the environment to detect knowledge faults with directed manipulation through experiments

The Learner-at-Heart strategy:

- Avoid deep nodes
- Prefer shallow nodes
- Avoid long plans
- Prefer short plans
- Prefer unreliable operators

The Planner-at-Heart strategy:

- Support main goal concord
- Avoid main goal protection violation
- Avoid main prerequisite violation
- Avoid irreversible operators
- Prefer reversible operators
- Prefer more reliable operators
- Avoid unreliable operators
- Prefer plans with fewer state changes
- Avoid plans with too many state changes
- Prefer operators that minimize state changes
- Avoid objects of very high protection
- Prefer objects of lower degree of protection
- Prefer least number of protected objects

Figure 2: Policies for two different experimentation strategies.

that lead to the missing knowledge. Our approach improves a planner's prediction accuracy and reduces the amount of unreliable action outcomes in several domains through the acquisition of new preconditions and effects of operators.

This work is applicable to a wide range of planning tasks, but there are some limitations. The state of the world must be describable with discrete-valued features, and reliable observations must be available on demand. Actions must be axiomatizable as deterministic operators in terms of those features. Another assumption is the absence of exogenous events or other agents in the environment that can change the state of the external world. Our work also assumes an initially incomplete knowledge base. Future work is needed to address other types of imperfections, including incorrectness of planning domain knowledge.

Acknowledgments

This work was done while the author was at Carnegie Mellon University. I would like to thank Jaime Carbonell, Tom Mitchell, Herb Simon, and Nils Nilsson for their suggestions and support throughout my thesis work. Thanks also to all the members of the PRODIGY group for many helpful suggestions. This research was supported by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597. The view and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of ARPA or the U.S. government.

References

Carbonell, Jaime G., Craig A. Knoblock, and Steven Minton. 1991. PRODIGY: An integrated architecture for planning and learning. In *Architectures for Intelligence*, ed. Kurt VanLehn. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cheng, Peter C-H. 1990. *Modelling Scientific Discovery.* PhD thesis, The Open University, Milton Keynes, England.

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

Gil, Y. Efficient domain-independent experimentation. In *Proceedings of the Tenth International Conference on Machine Leaning*, Amherst, MA. Morgan Kaufmann, 1993.

Gil, Y. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *Proceedings of the Eleventh International Conference on Machine Leaning*, New Brunswick, NJ. Morgan Kaufmann, 1994

Hammond, Chris J. 1986. *Case-based Planning: An Integrated Theory of Planning, Learning, and Memory.* PhD thesis, Yale University, New Haven, CN.

Huffman, Scott B., Douglas J. Pearson, and John E. Laird. 1992. Correcting imperfect domain theories: A knowledge-level analysis. In *Machine Learning: Induction, Analogy and Discovery.* Boston, MA: Kluman Academic Press.

Kedar, Smadar T., John L. Bresina, and C. Lisa Dent. 1991. The blind leading the blind: Mutual refinement of approximate theories. In *Proceedings of the Eight Machine Learning Workshop.* Evanston, IL.

Kulkarni, Deepak S. 1988. *The Process of Scientific Research: The Strategy of Experimentation.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Maes, Pattie and Rodney A. Brooks. 1990. Learning to coordinate behaviors. In *Proceedings of the Eight National Conference on Artificial Intelligence.* Boston, MA.

Mahadevan, S. and Connell, J. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence* 55(2-3):311–365, 1992.

Minton, Steve, Craig A. Knoblock, Dan R. Kuokka, Yolanda Gil, Robert L. Joseph, and Jaime G. Carbonell. 1989. PRODIGY *2.0: The Manual and Tutorial.* Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Rajamoney, Shankar A. 1993. The design of discrimination experiments. *Machine Learning*, 12(1/2/3), 1993.

Shen, W. M. Discovery as autonomous learning from the environment. *Machine Learning*, 12(1/2/3), 1993.