Explanation-Based Acquisition of Planning Operators

Geoffrey Levine and **Gerald DeJong**
{levine, dejong}@cs.uiuc.edu
Department of Computer Science
University of Illinois at Urbana

Abstract

Classical planning algorithms require that their operators be simple in order for planning to be tractable. However, the complexities of real world domains suggest that, in order to be accurate, planning operators must be complex. We demonstrate how, by taking advantage of background knowledge and the distribution of planning problems encountered, it is possible to automatically construct planning operators that are both reliable and succinct. The acquired operator is an encapsulated control loop that is specialized to best fit observed world behavior. Succinctness is achieved by publishing to the planner only those conditions required to succeed over the estimated distribution of problems. We demonstrate the acquisition of a context-appropriate "take-off" operator that can successfully control a complex flight simulator.

Introduction

In the classical planning framework, actions are specified in an operator format, by a set of preconditions that must hold before an action can be performed, and a set of effects that result. For real world domains, these operators typically represent some complex control problem of their own. Consider the blocks world, where we use a robotic arm to pick up and place blocks. The physical task of moving each block is non-trivial. A number of contingencies, such as motor failure or obstruction by an obstacle, could arise and potentially affect the applicability or outcome of our action. Thus, the nuances of the real world necessitate that, in order to be accurate, planning operators must be complicated.

However, in order for planning to be tractable, a planner requires that its operators be simple. Here we have a conflict. By simplifying operators, we sacrifice the reliability of our plans. But by specifying complex operators, we no longer have a means of solving our planning problem.

So, how can we simplify our operators without paying a costly price in reliability? Consider again the blocks world. Our stack action may fail if some obstacle obstructs the robotic arm, if a motor burns out, if a block is glued to the table, etc. We take a risk by leaving out any of these details from our operator definition. However, when we consider the distribution of problems that our planner comes across, the risk is unlikely to be uniform. Obstacles in the arm's path may be likely if the arm is used in a crowded environment, but unlikely otherwise.

Thus, there exists an opportunity to maintain both reliability and simplicity in our planning operators if we can identify those details of the world dynamics likely to become relevant within our distribution of problems and structure our operators accordingly. In this paper, we present a method to accomplish just this.

Overview

Standard classical planning proceeds as in Figure 1. A problem generator produces planning problems according to some fixed but unknown distribution. The planner takes in the problems, along with a library of operators, and outputs plans, which, when applied to the domain, result in some set of experiences. In this framework, the operators are designed by a domain expert and are immutable. By necessity, the domain expert has simplified the operators in order for the planning phase to be manageable but without any knowledge of the distribution of problems in which they will be applied. Thus, the specificity of the operators is likely misplaced. The operators may reference unimportant details of the domain, such as the possibility of motor failure when, in fact, our robotic arm is well maintained, or ignore important aspects, such as the presence of obstacles in the arm's path, if our tasks frequently take place in a crowded environment.

We supplement the planning process with a new *operator design* module, detailed in Figure 2. Instead of planning with an *a priori* fixed set of operators, specified by a domain expert, this module takes advantage of the observed world experiences to tailor operator definitions to the particular distribution of planning problems. The domain expert specifies only a general body of domain knowledge. The knowledge can be of varying specificities, and need not be consistent.

Three submodules make up the operator design module. An *explanation* submodule is used to associate observed world dynamics with a consistent causal model, calling on explanations from the knowledge base. A *calibration* mechanism associates precise numerical functions to the qualitative causal structure. Finally, the *publication* module assesses the capabilities of our operators against the distribution of planning problems, and produces a minimal sufficient set of operator definitions for use by the planner.

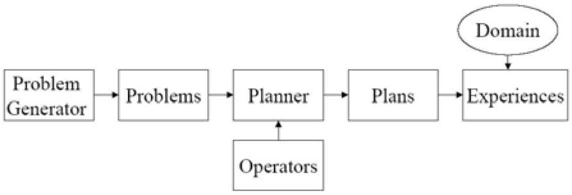

Figure 1: The Classical Planning Framework

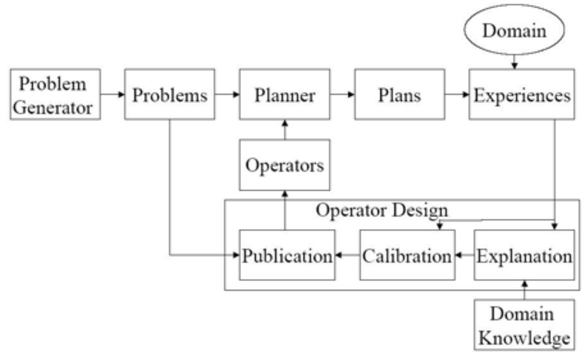

Figure 2: Addition of Operator Design Module

Explanation Given experiences in the world, the explanation module searches the knowledge base for the simplest consistent causal explanation of the observed phenomena. With a suitable subset of domain knowledge, the module constructs a qualitative *relationship graph*, a causal influence model associating predecessor states to action effects. As new data is experienced, it is tested for consistency against our current relationship graph. When inconsistencies are discovered, the current model is discarded as insufficient, and the online search identifies the simplest consistent alternative.

Calibration A qualitative relationship graph represents a family of numerical functions detailing the effects of an operator. The calibration module inputs this qualitative relationship graph and utilizes the quantitative data from our experiences to associate each underlying qualitative relationship with a consistent analytic relationship. The composition of these relationships, in turn, represents a family of predecessor state to action effect functions. We introduce a sound method of inference for accomplishing this task.

Publication After calibration of the relationship graph, the publication module weighs the possible effects of our actions against the estimated distribution of planning problems observed, and constructs a minimal set of capable operators for use by the planner. By referencing the distribution of planning problems, the publication module is able to unburden the planner by removing from the action schemas those conditions unlikely to become relevant.

Operators and Controllers

Consider again the blocks world domain, and the operator for unstacking a block. In the STRIPS planning language (Fikes and Nilsson 1971), the unstack operator is usually defined as follows.

```
Op(ACTION:  Unstack(b, x),
   PRECOND:  On(b, x) and Clear(b),
   EFFECT:  On(b, Table) and Clear(x)
            and !On(b, x))
```

Consider, though, how the unstack operator is implemented with a real robotic arm. Some controller, given block b as a parameter, completes the unstack operation by executing a series of arm motions:

1. Raise the robotic arm to be above block b.
2. Move the gripper horizontally to be directly above b.
3. Open the gripper.
4. Lower the gripper around block b.
5. Clamp the gripper around block b.
6. Lift the gripper slightly.
7. Move the gripper above an open space on the table.
8. Lower the gripper to the table.
9. Open the gripper, releasing block b on the table.

Notice that there are some internal parameters to this controller, such as the height to raise the arm, how tightly to clamp the gripper, where to place block b on the table, etc. If possible, we prefer to keep these decisions "under the hood," abstracted from the operational definition, as the more details the planner must observe, the more complex the planning process will be. If we are not concerned with the amount of time the controller takes, or the amount of battery power used, the planner can ignore these decisions without trouble. However, if the gripper is used for extensive amounts of time without recharging, we may find that our plans fail from time to time, when the battery runs out of energy. If this is the case, in order to plan reliably, we must include details such as the distance to move the gripper and the remaining battery power in the action schema.

In general, suppose our operator is implemented by some controller, c, with internal parameters, A_c. When instantiated from a state s $\in$ S with parameters a $\in$ A_c, the controller takes over until relinquishing control in a new state s'. Some function, T_c, describes the transition (T_c(s, a) = s'). We aim to learn, with knowledge and experience, the function T_c, such that the planner can confidently schedule actions within the problems of interest. For real-world tasks,

T_c will be quite complex. However, we exploit the fact that few distributions of problems will necessitate learning T_c precisely.

The basic effects of the unstack controller (those specified in the action schema above), will take very few examples to verify. However, the more fine grained effects, such as the energy consumption, the wear on the motors, etc, may take a large number of examples to determine, as they are complicated relationships dependent on a large number of factors (the height of the block stacks, the horizontal distance to an open area on the table, the weight of the block to be moved). If the energy consumption turns out to be irrelevant in the types of planning problems we see, though, we are perfectly happy for our planner to ignore these details.

Prior Knowledge

Given a controller, c, expertly specified domain knowledge is used to help describe T_c. However, for nontrivial tasks, this relationship can be quite complex, and very little can be said directly about the relationship. Fortunately, in many domains, the introduction of intermediate variables allows us to decompose T_c into a combination of simpler relationships, about which prior knowledge can be expressed, either qualitatively or, in some cases, quantitatively.

Variables

Relationships are expressed in terms of four types of variables:

- Initial State Parameters - Parameters of the state in which the controller is initialized
- Controller Action Parameters - Internal parameters of the controller
- Intermediate Variables - Observable parameters of the controller's execution path
- Final State Parameters - Parameters of the state in which the controller terminates

Relationships

Prior knowledge about functional relationships may be expressed in one of two forms, *analytic quantitative relationships* and *qualitative monotonic relationships*. Analytic relationships are exact numerical functions mapping from some set of variables to another. Qualitative monotonic relationships also specify a set of independent variables and a dependent variable, but only specify the sign of the partial derivative of the dependent variable with respect to each of the independent variables. For example, consider a relationship between cell phone antenna size (s) and distance from the transmission tower (d) to cell phone signal quality (q). Although we may not know the true function, we can be confident that, all other things being equal, signal quality cannot decrease with increasing antenna size, and cannot increase as the distance from the tower increases. That is, $\frac{\partial q}{\partial s} \geq 0$ and $\frac{\partial q}{\partial d} \leq 0$. Obviously, knowing a relationship analytically sidesteps a possibly complicated learning problem. As for qualitative monotonic relationships, the learning is still non-trivial.

Our body of knowledge contains relationships in which the dependent variable is either an intermediate variable or a final state parameter. It is not necessary for all statements to be consistent. That is, two knowledge statements can share a dependent variable but reference distinct sets of independent variables, or different qualitative/quantitative forms. Alternative statements like these provide the basis on which the explanation module evaluates alternative causal structures for the observed experiences. We also define the concept of a *null relationship*, which references a dependent variable but no independent variables. Intuitively, the null relationship signifies that the corresponding variable is irrelevant and unrelated to the world dynamics. We require that our body of knowledge include a null relationship for each intermediate variable/final state parameter to allow the system to conjecture that the variable may be insignificant.

Relationship Graphs

Given a body of knowledge about individual relationships, we introduce the notion of a *relationship graph*, a causal structure tying our initial state and controller parameter choice to the terminal state. Suppose our state S is represented as a factored set of components, $\{s_1, s_2, ..., s_n\}$, and our action representation A_c factors to $\{a_1, a_2, ..., a_m\}$. A relationship graph is a directed graph such that:

- Nodes correspond to variables, and are either independent or dependent:

  Independent nodes correspond to initial state parameters and controller action parameters. These nodes must have in-degree 0.

  Dependent nodes correspond to intermediate variables and final state parameters. These nodes can have in-degree ≥ 0.
- Directed edges indicate functional relationships. For all dependent nodes, there exists a function from the variables associated with the node's parents to the variable associated with the node, consistent with some statement in the relational prior knowledge.
- The graph is acyclic. This ensures that there are no circular dependencies.

Initial state nodes represent the state of the world when the controller is first instantiated. Final state nodes represent the changes performed by the operator. Not all of s_1 to s_n must be represented in the initial or final state nodes. Those factors unrepresented in the initial state nodes are assumed to be irrelevant to the operator effects. Those not in the final state nodes are assumed to be unaffected by the controller. Thus, the simplest possible relationship graph has no nodes at all, representing that the controller doesn't depend on the initial state and makes no changes to the world.

Relationship graphs are used to represent the function T_c. Each variable corresponding to a dependent node is bound as the dependent variable in some functional relationship. As there are no circular dependencies, assigning a value to each independent node defines a value for each dependent node, and subsequently a final state. Thus, the relationship graph utilizes intermediate variables to specify a mapping

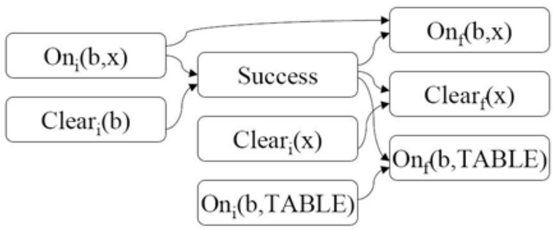

Figure 3: Relationship Graph for the Unstack operator

(s, a) to s'. As each of s_1, s_2, ..., s_n may appear twice in our relationship graph (once as an initial state node and once as a final state), we distinguish the two occurences with i and f subscripts, for initial state and final state respectively.

The unstack operator can be represented using the relationship graph in Figure 3. We introduce one intermediate variable, Success, to express whether the action is performed or not. The value of Success is dependent on the value of the preconditions (On_i(b,x), $Clear_i$(b)) and if true, defines the value of the relevant final state parameters (On_f(b,TABLE), $Clear_f$(x), On_f(b,x)). If Success is false, the final state parameters remain unchanged. One could imagine more complex relationship graphs describing energy consumed, motor wear, etc...

Explanation Module

The explanation module is the first submodule within the operator design module. In the flavor of explanation based learning (DeJong 2004), this module aims to produce, from our knowledge base, a consistent explanation for the observed experiences. Explanations take the form of relationship graph structures. In order to simplify the calibration and publication phases, and ultimately the published operator, when multiple relationship graph structures are consistent with the observed experiences, the simplest is chosen and provided as a base for calibration and publication to take place.

Consider taking a subset of relational statements from our knowledge base and evaluating the following three criteria.

- Consistency - For any variable, x, there exists at most one relationship referencing x as its dependent variable.
- Completeness - All intermediate variables and final state parameters referenced within the relational statements are themselves the dependent variable within one of the relationships.
- Acyclicity - Chaining forward from independent variables to dependent variables in our relationships never results in a cycle.

If a subset of knowledge satisfies these three criteria, it corresponds to a relationship graph, a potential causal explanation for the world dynamics. Furthermore, we define relationship graph g to be consistent with data set D if D is consistent with all of the corresponding relational knowledge used to construct g. That is, all data points must fall on the underlying quantitatively known relationships, and no pairs of data points can be arranged in a manner inconsistent with any of the qualitatively known relationships. The explanation module discards all relationship graphs that are inconsistent with the observed experiences, D. In order to select a particular consistent graph, we next present the concept of complexity.

Complexity

To define the notion of complexity on relationship graphs, we first define a partial ordering over relationships. For some intermediate variable or final state parameter x, consider all relationships citing x as the dependent variable. We can define a partial ordering on these relationships as follows. For relationships r and r', r $\leq$ r' if 1) r, r' are qualitative relationships, the set of independent variables for r is a subset of the independent variables for r', and the monotonic constraints on the independent variables of r are the same in r', or 2) r is the null relationship.

Define $r_{x,g}$ as the relationship used to describe x in graph g (possibly the null relationship). Given the partial ordering on relationships for each independent variable, we can define a new partial ordering, which we will call complexity, on relationship graphs as follows. For graphs g and g', g is less complex than g' if and only if g $\neq$ g', and, for all independent variables x, $r_{x,g} \leq r_{x,g'}$. In particular, the *null graph*, that composed entirely of null relationships, is less complex than all other graphs.

Given all possible relationship graphs, we can narrow our consideration to the consistent graphs that are minimally complex, those graphs that are both consistent with the observed experiences, and for which no other graph is both consistent and less complex. However, given the large number of potential relationship graphs, enumerating and evaluating every possible relationship graph is not an option. So, we define the following. Given graph g, the set of next most complex graphs, com(g) = {g' | g less complex than g' and $\nexists$ g" s.t. g less complex than g" and g" less complex than g'}. With this notion, we have a basis to define an incremental method for search among relationship graphs.

Search

We aim to find the simplest relationship graph capable of accurately describing the dynamics within our domain. Thus, the relationship graph search maintains a queue, Q, of candidate relationship graphs at all times, along with a singular simplest world model m. The search proceeds from simple to complex, and works as follows.

1. Q $\leftarrow$ {null graph}
2. m $\leftarrow$ first element in Q
3. While m consistent with D, Wait
4. Q $\leftarrow$ (Q $\cup$ com(m)) - {m}
5. Sort Q
6. Go to 2

Importantly, the search waits at step 3 until the planning agent's execution introduces new data that is inconsistent with m. As our goal is to find the least complex relationship graph consistent with the data, Q is to be sorted from least to most complex. However, complexity, as defined above, is only a partial order, so we will still have sets of relationship graphs that are incomparable. We define the following cost function, c(g) = the sum of independent variables in the qualitative relationships in g. c(g) is consistent with complexity in that g less complex than g' implies that c(g) $\leq$ c(g'), but it assigns a numeric value to all graphs, allowing comparison of all pairs of graphs. Q can be sorted based on c(g), inducing a uniform-cost search. More directed searches are possible through the inclusion of a heuristic function, which must estimate the number of amendments necessary for the relationship graph to become consistent with the observed data. One simple heuristic is h(g) = the number of qualitative relationships inconsistent with the observed data. The heuristic is admissible because all violated qualitative relationships must be supplemented with at least one more independent parameter to become consistent.

The relationship graph m, the simplest consistent graph, is the output of the explanation phase, and the basis on which calibration proceeds.

Calibration Module

The second component of our system is the calibration module. This module takes as input the qualitatively consistent relationship graph, m, produced by the explanation module, and resolves it with the observed data, in order to assign it an overall functional relationship S $\times$ $A_c \rightarrow$ S. This necessitates assigning analytic functions to those relationships represented qualitatively. The goal is to as accurately as necessary represent T_c, but in general the conjunction of our prior knowledge and experiences will not be sufficient to isolate a single consistent function. There are several ways to resolve this problem. An appealing option is to assume some model underlies each of the qualitative relationships and use the observed data to regress to a best fit. Unfortunately, approximating the true function means that we must give up any firm guarantees of accuracy. Thus, we introduce the following notions, and assign our relationship graph a range of possible functions.

Let P denote the prior knowledge underlying m. Without knowing T_c exactly, taking into account the experiences in our domain, D, we can define a set of *Consistent Transition Functions*, CT_c(P, D) = $\{T_c$: S $\times$ $A_c \rightarrow$ S $|$ T_c consistent with P, D$\}$. Given an initial state, s and an action a, we can define the set of possible successor states for controller c as follows. $succ_c$(s, a) = $\{$s' $\in$ S $|$ $\exists$ $T_c \in CT_c$(P, D) T_c(s, a) = s'$\}$. In this manner we can be guaranteed that the true T_c(s, a) is in $succ_c$(s, a), as long as P is correct. In order to accomplish this, we introduce the following inference procedure.

Function Learning with Qualitative Knowledge

Consider again the cell phone domain, and suppose we aim to learn the function between antenna size (s) and signal quality (q), assuming a fixed distance from the tower. We gather data by attaching antennas of different sizes, and recording the resulting signal quality. Suppose our first data point is s = 5 cm, q = 50 dB. Without any additional knowledge about the world function f(s) = q, this data point is rather meaningless. However, with the qualitative knowledge that the function f(s) = q is nondecreasing, our one data point makes a strong set of claims. We can immediately rule out all points where s $>$ 5 cm, q $<$ 50 dB and those where s $<$ 5 cm, q $>$ 50 dB, as they are inconsistent with the conjunction of our prior knowledge P and data D. As we gather more data, we can rule out more regions of the (s, q) space, maintaining a smaller and smaller set of (s, q) points consistent with P, D.

When more than one independent variable is involved, function learning follows the same principle. Consider varying both antenna size (s) and distance to transmission tower (d). If our first data point is s = 2 cm, d = 1 km, q = 40 dB, we can immediately rule out all points where s $>$ 2 cm, d $<$ 1 km, q $<$ 40 dB and those for which s $<$ 2 cm, d $>$ 1 km, q $>$ 40 dB as inconsistent with P, D.

In general, suppose we have a function, f, mapping from independent variables $x_1, x_2, ..., x_n$, to dependent variable y, and we know that the function either monotonically increases or decreases with each x_i. Given a data point of the form x_1 = a_1, x_2 = a_2, ..., x_n = a_n, y = b, rule out those points, x_1 = c_1, x_2 = c_2, ..., x_n = c_n, y = d, where either:

1) d $>$ b, $\forall i$ (($\frac{\partial y}{\partial x_i} \geq 0$, $c_i \leq a_i$) or ($\frac{\partial y}{\partial x_i} \leq 0$, $c_i \geq a_i$))

2) d $<$ b, $\forall i$ (($\frac{\partial y}{\partial x_i} \geq 0$, $c_i \geq a_i$) or ($\frac{\partial y}{\partial x_i} \leq 0$, $c_i \leq a_i$))

Theorem 1 *The functional learning mechanism is sound*

Proof Sketch: We outline the proof for line 1). The proof for line 2) follows analogously. Suppose f(a_1, a_2, ..., a_n) = b. If $\forall i$ (($\frac{\partial y}{\partial x_i} \geq 0$, $c_i \leq a_i$) or ($\frac{\partial y}{\partial x_i} \leq 0$, $c_i \geq a_i$)), consider the effect on f(x_1, x_2, ..., x_n) of changing each x_i from a_i to c_i. For each i, the above constraint requires that the function output can only decrease if we fix all other inputs, but change the ith input from a_i to c_i. If we do this for each i, 1 to n, the function output must be non-increasing, and thus f(c_1, c_2, ..., c_n) $\leq$ b.

Relationship Graph Inference

Utilizing the mechanism above for function learning and the relation graph structure input from the explanation phase, graph m, the following method allows us to propagate initial states and action choices to a possible range of final states, defining $succ_c$(s, a). Given prior knowledge P, data D, initial state s $\in$ S, and action choice a $\in$ A_c:

1. Mark all nodes in m as unprocessed.
2. Assign the values from s and a to the associated initial state and action nodes of m, mark these nodes as processed.
3. Select a dependent node, n, from m, such that n's parents are all marked as processed.
4. Assign a range of values for n based on the range of values associated with its parents, consistent with P, D.
5. Mark n as processed.

6. If unprocessed nodes exist in m, return to step 3.
7. Define $succ_c$(s, a) as the union of ranges of values from the final state nodes of m.

Theorem 2 *The above procedure will return a range of values for the final state that includes T_c(s, a)*

Proof Sketch: As the graph is acyclic, a sufficient dependent node must always exist at step 3. Because the graph has a finite number of nodes, the inner loop will repeat a finite number of times, and the procedure will terminate. Furthermore, as the functional learning mechanism is sound by Thm 1, the range of values produced for node n at step 4 must include all possible values associated with the values of n's parents. Starting with the first dependent node and stepping forward, by induction, the range of values associated with each dependent node must include the corresponding variable's actual value, defined by s, a. In particular, the actual value of the final state associated with s, a must be included in the range of values associated with the final state nodes.

So, while sacrificing the benefit of simplicity in isolating a particular function, we maintain the guarantee of accuracy. Of course, the more data we acquire, the tighter our range of final states will be. The key is, at some point, we will acquire enough data that we can guarantee success within the problems of interest. This task is handled in the final stage of operator design.

Publication Module

In the calibration phase, each action, state pair is associated with a range of final states, but we have yet to construct operators for use by our planner. The publication module takes as input the calibrated relationship graph, and compiles a set of one or more such operators. The general goal of publication is to supply the planner with a set of simple operators capable of satisfying the planner's needs in the given distribution of problems.

When T_c is complex, many potential operators are possible, corresponding to the varying effects of particular state, action pairs. However, many distributions of planning problems are solvable with a limited set of these operators. In these cases, publishing operators for all such contingencies unnecessarily complicates the planning process. Instead, the publication module finds a small set of operators, consistent with T_c, that are sufficient for solving the planning problems encountered. While we do not offer a general solution for operator publication, we demonstrate in the implementation section how, given a particular problem, this can be efficiently accomplished.

The Flight Domain

We demonstrate our method in a complex flight simulator domain. Flight is an interesting domain for planning because, despite the complex world dynamics and the large number of control settings to manage, a suite of simple flight operators (takeoff, ascend, turn, fly straight, descend, land), can be used to create most flight plans of interest. We focus on the takeoff operator. We are provided with a stationary plane located on a runway. We must select control settings such that the plane will takeoff and ascend in such a manner to safely clear an obstacle located in the distance.

The relevant factors in the initial state are the presence of the plane on the runway (denoted with predicate OnRunway), the distance and height of the obstacle (D, H), as well as the time that the controller takes over (T_i). Action choices are made up of a (V, F, B) tuple. V (velocity) is the horizontal speed to which we accelerate and which we maintain during ascent. It ranges between 75 and 100 knots. F ranges from 0 to 1, and denotes how far to extend the wing flaps during takeoff. B is the back taxi distance, the distance that we taxi our plane backwards down the runway before we begin the takeoff acceleration, and ranges from 0 meters to 500 meters. The relevant factors in the final state include Clear, a predicate indicating that our plane is clear of all obstacles in the vicinity, T_f, the time when the takeoff controller terminates, and OnRunway, which is false after a successful takeoff.

The operator is implemented as follows. If $B > 0$, the plane turns around and taxis B meters backwards down the runway before turning forward again. Flaps are then directly set to the value of F (via a knob in the cockpit). The throttle is set to its maximum value, and the plane begins acceleration. When the plane reaches velocity V, a PD controller is switched on to maintain the present velocity by controlling the elevator (the throttle remains at maximum power). A separate PD controller is used to maintain the heading of the plane down the runway. Control is relinquished when the plane reaches the horizontal position of the obstacle.

The controller choices lead to an interesting set of flight dynamics. Increasing V reduces the amount of time needed to reach the obstacle. However, flying at a higher velocity necessitates pointing the nose of the plane down a bit to maintain a high level of thrust. This reduces our ascent angle, and thus may lead to crashing into the obstacle. Extending the flaps increases the chord of the wing, and thus produces more lift for a given velocity, meaning that the plane lifts off of the ground earlier with flaps than without. However, extending the flaps increases the drag on the plane, causing the plane to struggle to maintain velocity, meaning it must nose down and fly at a lower ascent angle after liftoff. Finally, increasing B increases our distance from the obstacle, thus allowing us to clear taller obstacles with a given (V, F) setting. As this maneuver is slow, though, backtaxiing long distances increases the time duration of the takeoff operator substantially, but may be necessary for clearing some obstacles.

Prior Knowledge

As demonstrated above, the effect of setting the controller parameters on the final state is not immediately clear as the state and action parameters interact in complex ways. Thus, we specify knowledge of varying levels of specificity. In order to specify fine-grained knowledge, we introduce the following intermediate variables:

- AS (Ascent Slope), the slope that the plane maintains after lifting off.
- LD (Liftoff Distance), the amount of ground (meters) cov-

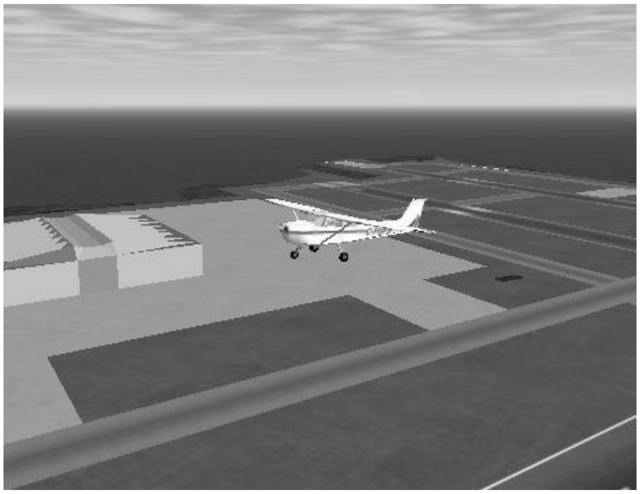

Figure 4: Taking off in FlightGear

ered during acceleration before the plane reaches its goal velocity and lifts off.

- BT (Back Taxi Time), the amount of time (seconds) necessary to back taxi the specified distance.
- LT (Liftoff Time), the amount of time (seconds) needed for the plane to accelerate to its goal velocity and lift off.
- AT (Air Time), the amount of time (seconds) that the plane is airborne before reaching the obstacle.
- TT (Total Time), The total amount of time (seconds) used by the operator to reach the obstacle from its initial state, sitting on the runway.

Given these variables, we specify a body of knowledge, containing knowledge of varying specificities. The complete body of knowledge is listed in the appendix.

Environment

To test our method, we use the freely-available open-source flight simulator FlightGear (Olsen 2005) (www.flightgear.org). All trials are performed using the default plane, the Cessna 172. Our operator definition module and control loops are implemented externally and communicate with FlightGear via a socket connection.

Problem Distributions

We demonstrate our method on problem distributions of varying difficulties to characterize the effect this has on the operator design module. The two distributions are:

- Easy Distribution - In this distribution OnRunway$_i$ = True, Clear$_i$ = False, T$_i$ = 0, D is distributed uniformly in distance from 3 to 4 kilometers, and the height H is distributed uniformly from 150 to 200 meters. These problems are all "easy" in that all obstacles can be cleared in minimum time by varying just the V parameter, and defaulting F and B to their minimum values. That is, none of the obstacles are either close enough or tall enough to require the use of back taxiing or flaps.

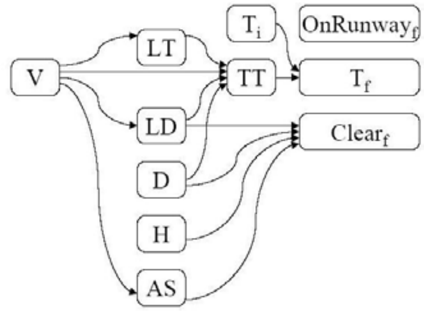

Figure 5: Graph g$_1$, based on statements 1, 3, 5, 6, 7, 9 from the appendix

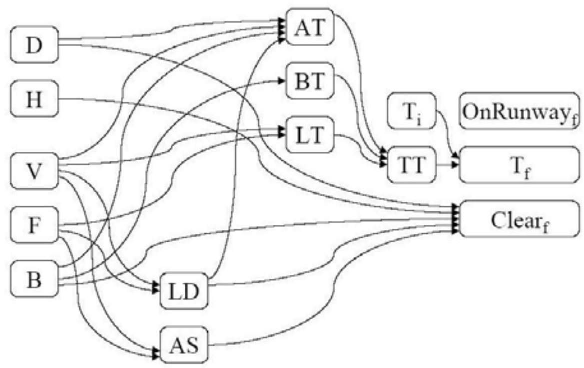

Figure 6: Graph g$_2$, based on statements 2, 3, 4, 6, 7, 8, 9, 10 from the appendix

- Hard Distribution - Again, OnRunway$_i$ = True, Clear$_i$ = False, T$_i$ = 0, but D is distributed uniformly in distance from .5 to 4 km, and H is distributed uniformly from 0 to 400 meters. In some of these initial states, the obstacle is very close and/or very tall, necessitating that we either back taxi in order to gain more space to ascend, or when the obstacle is close but short, use flaps to benefit from the shorter takeoff roll.

Search

Many potential relationship graphs are derivable from the knowledge base presented in the appendix. The explanation search procedure starts with the null graph, g$_n$, and moves towards more complex graphs as the simpler models are found to be inconsistent with the observed world dynamics. The details of the search procedure are dependent on the specific sequence of problems encountered, but the search always terminates with the simplest relationship graph consistent with the observed data.

For each distribution, experiences are obtained as follows. An initial state is drawn at random. An expert specifies an action setting (V, F, B) corresponding to the minimum time to successfully clear the obstacle, or at random if no such action setting exists. The takeoff controller is instantiated

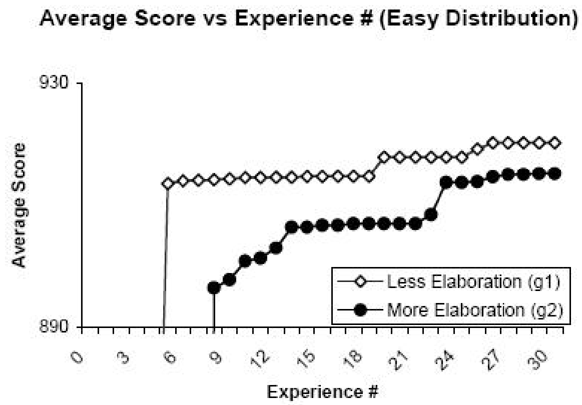

Figure 7: Learning Rate on Easy Distribution

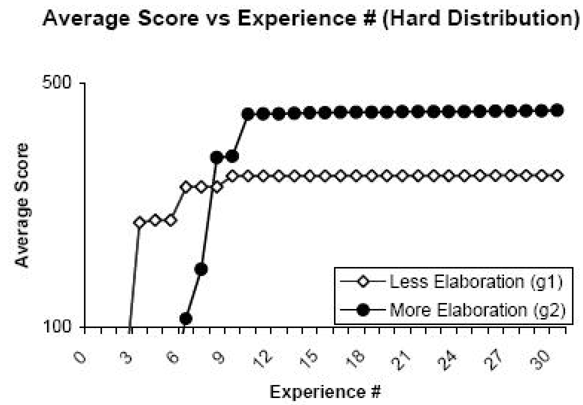

Figure 8: Learning Rate on Hard Distribution

in the world with the corresponding action setting, and the resulting experience is added to the data set. After each trial, the search procedure is woken up, and the current model, m, is tested for consistency.

For the easy distribution of problems, the search terminates with graph g_1 pictured in Figure 5. The relationship graph implies that OnRunway_f is always false, while T_f and Clear_f can be traced back as functions of the initial state/action parameters. Because all problems can be solved by varying just the value of the velocity parameter, V, the expert never entertains the possibility of changing the F and B settings from their default values, and thus the search never complicates the relationship graph with their unobserved influence on the world dynamics. Of the underlying relationships, three are qualitative and must be calibrated. Each of these relationships has one independent variable, velocity in all cases, and thus $c(g_1) = 3$.

For the hard distribution, the expert suggests non-zero F and B settings for the difficult initial states. Varying these controller parameters excites world dynamics that are inconsistent with g_1, and thus it is eventually rejected. Ultimately, a more elaborate graph, g_2, shown in Figure 6, is discovered and found to be consistent with the experiences. Based on our knowledge, there are other derivable relationship graphs that are also consistent with the world dynamics. However, graph g_2 has the lowest complexity measure ($c(g_2) = 7$), and is found first.

Learning Rate and Performance

Next we demonstrate the rate of learning accomplished by the calibration procedure. The calibration module takes as input a relationship graph structure, and assigns its qualitative relationships quantitative meaning in order to estimate $T_c(s, a)$. However, the rate at which we learn T_c depends heavily on the complexity of the graph. T_c is estimated based on the composition of all relationships making up the graph, and thus, graphs with many qualitative relationships require more data than those with with few. In order to characterize this rate of learning, we define the following quantities:

- Score - In the learning framework, we assign each experience a score. In this case, we define the score as 1000 - T_f if Clear_f is true, and 0 otherwise. Thus, we aim to clear the obstacle in the *minimum* amount of time.
- Best Guaranteed Score - After calibration, each initial state/action pair is associated with a range of final states, which can be translated into a range of scores. Given an initial state, for some action the lower bound on this range is the greatest. As selecting the corresponding action guarantees obtaining a score at least this high, we call this value the best guaranteed score. As more is learned about the qualitative relationships making up the relationship graph, these ranges will become more precise, and so we can expect to see the best guaranteed score improve.
- Dominated Actions - Again, given an initial state, every action can be associated with a range of possible scores. For some actions, the upper bound on this corresponding range will be less than the best guaranteed score. That is, selecting this action is guaranteed to result in a lower score than the action associated with the best guaranteed score. We will call these action choices dominated.

We first demonstrate the rate of learning on the easy distribution of problems. We consider as input into the calibration module graphs g_1 and g_2. A training set of 30 initial states are drawn independently at random from the distribution, as is a test set of 30. Training proceeds as follows. For each training state, each action is associated with a range of scores, as described above. An action is chosen uniformly from those that are not dominated, and applied in the simulator, resulting in an experience, which is added to our data set. After each training experience is acquired, the current data set is used by the calibration module to associate each of the test cases with a best guaranteed score, as described above. The mean of these scores at each training step is charted in Figure 7.

For both graphs, the data points start below the chart as the calibration mechanism has trouble guaranteeing that any control parameter setting will successfully clear any of the obstacles. As graph g_1 has fewer qualitative relationships than g_2, fewer experiences are needed to start seeing guarantees of success over the test set. Furthermore, as g_1 is capable of representing the best action for each state in the easy distribution, without having to calibrate qualitative relationships related to the flaps and back taxi settings, g_1 learns the optimal action choices and their associated scores faster than g_2.

The same procedure is followed with the hard distribution of problems in Figure 8. As in the easy distribution, the simplicity of graph g_1's underlying relationships give it a head start in learning. However, its inability to find ways to clear some of the more difficult obstacles means that its average guaranteed score levels off at a sub optimal value. Graph g_2, on the other hand, although slower to learn at first, is able to eventually find solutions to some of the more difficult problems that g_1 is incapable of representing, and ultimately surpasses g_1 in performance.

Operator Publication

The publication module inputs the calibrated relationship graph, and determines a minimal sufficient set of operators for the encountered planning problems. After calibration, our relationship graph is associated with a function mapping $S \times A \rightarrow S$. Alternatively, we could consider our calibrated relationship graph to represent a function $A \rightarrow (S \rightarrow S)$. That is, bounding the controller parameters, we have a function mapping from the relevant parameters of the initial state (T_i, D, H), to the relevant parameters on the final state ($OnRunway_f$, $Clear_f$, T_f). This function is easily expressible and can be formalized as a planning operator. Thus, our publication module will consider operators corresponding to each (V, F, B) setting.

We demonstrate the publication module in the following manner. Given the calibrated relationship graph, g_2, and 30 initial states generated from the hard distribution, we vary a real-valued time constraint parameter C. For an initial state i $\in$ S and an operator o, we say that o satisfies i if o guarantees successfully clearing the obstacle within time C. For each value of C, the publication module generates a minimum set of operators capable of satisfying the maximal number of planning problems.

When C is very large, for example 300, only one operator is acquired because we can safely back taxi the full distance without running out of time on any of the obstacles. The corresponding operator has preconditions

```
1) H < (D - 88) * .101 and D < 7700
```

where all values are in meters. The first condition ensures that our trajectory will clear the obstacle. The second guarantees that we will reach the obstacle in the allotted time.

As C shrinks, the system finds that back taxiing the maximum distance no longer allows us to reach the most distant obstacles in time, sparking the generation of a second operator which does not back taxi. At C = 150, we have two operators, with the following preconditions.

```
1) H < (D - 88) * .101 and D < 2000
2) H < (D - 802) * .101 and D < 5000
```

By forgoing back taxiing, the second operator is able to cover a greater distance in the allotted time. However, the trajectory is shallower, meaning that obstacles must be shorter and/or more distant.

Decreasing C even more results in another operator published to handle close and short obstacles that necessitate using flaps. When C = 110, the following three operators are constructed.

```
1) H < (D - 710) * .101 and D < 2200
2) H < (D - 850) * .101 and D < 3700
3) H < (D - 800) * .049 and D < 4000
```

As C decreases further, fewer and fewer of the obstacles become clearable in the given time constraint, culminating with no clearable obstacles, and thus no published operators at C = 40.

Related Work

Much work relating machine learning to planning involves learning control knowledge to guide the search for plans (Fikes, Hart, and Nilsson 1972) (Minton 1988). Several works have studied the problem of learning action schemas from examples in deterministic planning domains, including the LIVE (Shen 1994), EXPO (Gil 1994), TRAIL (Benson 1996), LOPE (Garcia-Martinez and Borrajo 2000), and GIPO (McCluskey, Richardson, and Simpson 2002) systems. These works use statistical and/or inductive techniques, but do not take advantage of expertly specified domain knowledge. Fewer works have focused on learning planning rules for nondeterministic domains. (Schmill, Oates, and Cohen 2000) clusters similar experiences and induces decision trees that model the dynamics of native actions. (Pasula, Zettlemoyer, and Kaelbling 2004) learns STRIPS-like probabilistic planning rules with several levels of search.

Reinforcement learning techniques (Sutton and Barto 1996) (Kaelbling, Littman, and Moore 1996) use reward and punishment to train an agent to complete a task. The general framework allows for goal directed learning without specifying how the goal is to be accomplished. However, without prior knowledge to direct the learning procedure, learning by reinforcement often necessitates vast amounts of training experiences.

A number of previous works have applied learning to complex flight domains. (Benson 1996) applies his work within the Silicon Graphics, Inc. flight simulator. PEGASUS (Ng and Jordan 2000) has been applied to the task of autonomous helicopter flight (Ng et al. 2004). (Camacho 1998) and (Issac and Sammut 2003) use records of human piloted flights in order to learn behavioral clones capable of flying precise maneuvers. Our approach is unique in applying EBL with qualitative domain statements.

Conclusion and Future Work

We supplement the classical planning framework with an operator design module. This module reacts to the observed world experiences and tailors planning operators accordingly. With the benefit of prior knowledge, the module searches for the simplest causal explanation of the world dynamics, calibrates it numerically, and generates one or more operators for use in planning. In a complex flight domain, we detail the operator design search and demonstrate the benefit in having a causal structure that matches the complexity of the problem distribution.

Our current research includes extending our method to stochastic domains through the use of alternative function approximators and suitably adjusted domain knowledge. Furthermore, we assume that our expertly designed controllers are sufficient for the problems encountered. An interesting step would be to use the domain knowledge to guide the construction of complex controllers from simple building blocks.

Acknowledgements

This material is based upon work supported in part by the Information Processing Technology Office of the Defense Advanced Research Projects Agency under award HR0011-05-1-0040. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Appendix - Takeoff Domain Knowledge

We include the following relational statements in our knowledge base, along with the null relationship for all intermediate and final state variables. Lines with qualitative knowledge specify relationships corresponding to all subsets of the independent variables. For example, statement 6 represents 4 qualitative relationships corresponding to each of {}, {V}, {F}, {V, F}.

1. $\text{Clear}_f = f_{Clear}(\text{D, H, AS, LD}) = \{$true if (LD + D) * AS $>$ H, false otherwise$\}$.
2. $\text{Clear}_f = f_{Clear}(\text{B, D, H, AS, LD}) = \{$true if (B - LD + D) * AS $>$ H, false otherwise$\}$.
3. $T_f = f_T(T_i, TT) = T_i + TT$.
4. $TT = f_{TT}(BT, LT, AT) = BT + LT + AT$.
5. $TT = f_{TT}(V, D, LT, LD) = (D - LD) * c / V + LT$. c is a conversion factor.
6. $AS = f_{AS}(V, F)$, $\frac{\partial AS}{\partial V} \leq 0$, $\frac{\partial AS}{\partial F} \leq 0$.
7. $LD = f_{LD}(V, F)$, $\frac{\partial LD}{\partial V} \geq 0$, $\frac{\partial LD}{\partial F} \leq 0$.
8. $BT = f_{BT}(B)$, $\frac{\partial BT}{\partial B} \geq 0$.
9. $LT = f_{LT}(V, F)$, $\frac{\partial LT}{\partial V} \geq 0$, $\frac{\partial LT}{\partial F} \leq 0$.
10. $AT = f_{AT}(V, B, D, LD) = (B - LD + D) * c / V$.

References

Benson, S. 1996. Learning Action Models for Reactive Autonomous Agents. Ph.D. diss., Dept. of Computer Science, Stanford Univ.

Camacho, R. 1998. Inducing Models of Human Control Skills. In *Proceedings of the Tenth European Conference on Machine Learning,* 107-118.

DeJong, G. 2004. Explanation-Based Learning. *Computer Science Handbook*, 68.1-68.18. Chapman Hall.

Fikes, R., Hart, P. Nilsson, N. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* 3:251-288.

Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189-208.

Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *Proceedings of the Eleventh International Converence on Machine Learning*, 87-95.

Isaac, A., and Sammut, C. 2003. Goal-directed Learning to Fly. In *Proceedings of the Twentieth International Conference on Machine Learning*, 258-265.

Kaelbling, L., Littman, M., and Moore, A. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4:237-285.

McCluskey, T.L., Richardson, N.E., and Simpson, R.M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*, 121-130.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers.

Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger E., and Liang, E. 2004. Inverted Autonomous Helicopter Flight via Reinforcement Learning. In *International Symposium on Experimental Robotics*.

Ng, A., and Jordan, M. 2000. PEGASUS: A Policy Search Method for Large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 406-415.

Oates, T., and Cohen P.R. 1996. Searching for Planning Operators with Context-Dependent and Probabilistic Effects. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 865-868.

Olsen, C. 2005. *Introduction to FlightGear.* http://www.flightgear.org/introduction.html.

Pasula, H., Zettlemoyer, L., and Kaelbling, L. 2004. Learning Probabilistic Relational Planning Rules. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 73-82.

Garcia-Martinez, R., and Borrajo, D. 2000. An Integrated Approach of Learning, Planning, and Execution. *Journal of Intelligent and Robotic Systems* 29:47-78.

Schmill, M.D., Oates, T., and Cohen, P.R. 2000. Learning Planning Operators in Real-World, Partially Observable Environments. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 246-253.

Shen, W.-M. 1994. Discovery as Autonomous Learning from the Environment. *Machine Learning* 12:143-156.

Sutton, R.S, and Barto, A. 1996. *Reinforcement Learning.* Cambridge, MA: MIT Press.