

we will look to extend the representations and techniques in this work to accommodate action descriptions that include conditional effects.

A plan is a sequence of actions that is intended to achieve a goal when executed in an initial state. Executing an action in a state in which all of its preconditions are true will produce a new state in which all of its effects are true and all other fluents remain unchanged. Executing an action with any false preconditions has no effect on the state.

In order to focus on issues related to incomplete action models, we do not address uncertainty about the state in this paper. Instead, we assume that the planner is given a complete state of the world, s_0 . However, this does not sidestep any fundamental issues because executing actions with incompletely modeled effects creates uncertainty about the state of the world.

In our discussion, we will often need to reason about whether a plan will be successful given the information contained in D . We define $achieves(D, s_0, g, p)$ as returning true iff the goal g is true in the state that results from simulating the execution of plan p from state s_0 assuming that D is correct and complete.

Representing incomplete action descriptions

We now present a language for supplementing D with statements about the completeness of action descriptions. Our approach follows the use of locally closed-world (LCW) statements that are used to overcome incomplete state information. In that setting, an LCW statement expresses limited knowledge of the state, such as that all the files that exist in a particular directory are known by a software agent (Golden, 1998; Babaian & Schmolze, 2000).

In our work, an LCW statement expresses limited completeness of action descriptions. These statements are defined in terms of three predicates: *DoesNotRelyOn*, *DoesNotMakeTrue*, and *DoesNotMakeFalse*. The truth value of these predicates is directly specified by the user; no inference is performed.

The statement $DoesNotRelyOn(a, x)$ asserts that action a does not have precondition x or $\neg x$ in D_{true} , where x is an atom. We define $CompletePreconditions(a)$ as a shorthand for the formula:

$$\forall x. x \notin preconditions(a) \supset DoesNotRelyOn(a, x)$$

which is equivalent to explicitly stating that the completeness of preconditions assumption is valid for a .

The statement $DoesNotMakeTrue(a, x)$ asserts that a does not have effect x in D_{true} , where x is a literal. The statement $DoesNotMakeFalse(a, x)$ asserts that a does not have effect $\neg x$ in D_{true} . We define $CompleteEffects(a)$ as a shorthand for the formula:

$$\forall x. x \notin effects(a) \supset DoesNotMakeTrue(a, x) \wedge DoesNotMakeFalse(a, x)$$

which is equivalent to explicitly stating that the completeness of effects assumption is valid for a .

There is a set of potential LCW statements that warrant special consideration. If an action a maintains (i.e., does not clobber) a listed precondition x , it is possible that x will not

be listed as an effect (since it is implied by the completeness of effects assumption). However, in most domains, $DoesNotMakeFalse(a, x)$ can safely be assumed to be true. Thus, our implementation can, optionally, automatically add such frame axioms to the user-specified LCW statements.¹

Plan selection

A *plan selection problem* is a 5-tuple (g, C, s_0, D, L) , where g is a goal, C is a set of candidate plans, s_0 describes the initial state, D is a potentially incomplete action model, and L is a set of locally closed world statements. The ideal solution to this problem is to find the plan that will achieve its goal given the actual action descriptions. More precisely, the objective is to find $c_i \in C$ so that $achieves(D_{true}, s_0, g, c_i)$. The quality of a plan selection algorithm can be measured by how frequently it chooses a plan that actually achieves g . Another measure is how often the plan selection algorithm indicates that no plan should be executed if, in fact, no plan in C will achieve g given D_{true} .

Algorithm and Analysis

We consider the following four types of risks, each of which is a potential source of execution failure, for a plan composed of actions a_1, \dots, a_n . Without loss of generality, we encode the initial state as an action a_0 with no preconditions and effects s_0 and encode the goal g as action a_{n+1} which has preconditions g and no effects.

- $POSSCLOB(a_i, x) ::$ action a_i might have the effect $\neg x$, and there exists action a_j , for $i < j$ that has precondition x in D and no action between a_i and a_j has effect x in D .
- $PRECOPEN(a_i) ::$ action a_i might have an unlisted precondition that will not be true when a_i is executed in a_0, \dots, a_n .
- $PRECFALSE(a_i, x) ::$ a_i has a precondition x in D which will be false when executed in a_0, \dots, a_n , according to D .
- $HYPOTHEZEDEFFECT(a_i, x) ::$ the correctness of the plan relies on an effect x of a_i that is not listed in D , but might be part of D_{true} . This means that x is consistent with the description of a_i in D , but there is no evidence to support the hypothesis that x is part of the description of a_i in D_{true} .

The first two types of risks correspond to relying on knowledge implied by completeness assumptions, i.e., that D is a good approximation to D_{true} . In contrast, the latter two risks rely on the incompleteness of the model in order to justify plans that would fail if $D = D_{true}$. For example, an action with a $PRECFALSE(a_i, x)$ risk cannot successfully execute unless a previous action has an effect x that is not in D , or the action that negates x has an unlisted precondition.

If no plan justified by the action descriptions in D can achieve goal g then it is worth considering plans with other possible risks. For example, it seems preferable to execute a plan with one hypothesized effect than one with five such

¹This can be done trivially by a pre-processor.

```

FINDRISKS ( $\langle a_1, \dots, a_n \rangle, g, L$ )  $\equiv$ 
  RiskSet  $\leftarrow \emptyset$ 
  Unsupported  $\leftarrow g$ 
  for  $i = n$  to 1
    Supported  $\leftarrow$  Unsupported  $\cap$  EFFECTS( $a_i$ )
    Supported  $\leftarrow$  Supported  $\setminus$  PRECONDITIONS( $a_i$ )
    Unsupported  $\leftarrow$  Unsupported  $\setminus$  Supported
    for all literals  $x$  in Unsupported
      if DoesNotMakeFalse( $a_i, x$ )  $\notin L$ 
        RiskSet  $\leftarrow$  RiskSet  $\cup$  { POSSCLOB( $a_i, x$ ) }
      if CompletePreconditions( $a_i$ )  $\notin L$ 
        RiskSet  $\leftarrow$  RiskSet  $\cup$  { PRECOPEN( $a_i$ ) }
    Unsupported  $\leftarrow$  Unsupported  $\cup$  PRECONDITIONS( $a_i$ )
  return RiskSet

```

Figure 1: Finding risks in a provably correct plan.

effects.² However, these plans must be weighed against the alternative of doing nothing, which can be the best option if the chance of achieving a goal is small. Assessing these options would be more practical within a probabilistic framework.

The function FINDRISKS, shown in Figure 1, produces the set of risks of a plan $c = a_1, \dots, a_n$, a goal g , and a set of locally closed world statements L . The algorithm requires $O(nm)$ time, where n is the length of the plan, and m is the number of literals in D . The pseudo-code is streamlined by assuming that c is a provably-correct plan in D ; therefore, there is no need to check for hypothesized effects, false preconditions, or that g is achieved.³

FINDRISKS requires one pass over the plan, working backwards from the last action. During processing, it keeps track of the set of literals that must eventually be achieved by an earlier action or be true in the initial state of the world. Each action a_i may provide support for some of these literals, namely the literals that are made true by the action. For each remaining unsupported literal x , the set of LCW statements L is checked to see if a_i might clobber x ; if so, POSSCLOB(a_i, x) is added to the result. Also, L is checked to see if a_i 's preconditions are completely modeled; if not, PRECOPEN(a_i) is added to the result. It might seem strange that PRECONDITIONS(a_i) is not checked to be true; however, this is not needed since we assume that the input plan is provably correct in D .

Analysis

We now discuss the implications of risks. We show that a plan without risks cannot fail. We then describe a tractable method for identifying a subset of risks which imply the possibility of plan failure.

Roughly speaking, we say that it is possible for a plan to fail if there exists a possible world that is consistent with the

²How and why a planner would hypothesize such effects are beyond the scope of this paper.

³A more general version of this algorithm exists that efficiently finds these risks as well. When simulating the execution of the plan using D , if an action's preconditions are false, we treat it as having no effects and add a PRECFALSE risk to *RiskSet*.

information given the planner in which p does not achieve its goals. More precisely, we mean there exists some D' such that $D \subseteq D'$, D' is consistent with the locally-closed world statements L , and \neg achieves(D', s_0, g, c).

Determining if a plan will succeed involves reasoning over all possible states that might occur in the execution of the plan, which is exponential in the length of the the plan if there is uncertainty in the outcome of the actions. Under our assumption of a correct but incomplete model, any plan that can fail in D_{true} will have at least one risk. The following theorem states that a plan that has no risks and will succeed given action model D will also succeed given any action model that is a superset of D (and a subset of D_{true}). As a corollary, the plan will succeed given D_{true} .

Theorem: if $D \subseteq D' \subseteq D_{true}$ and FINDRISKS(c, g, L) contains no risks and achieves(D, s_0, g, c) then achieves(D', s_0, g, c).

Proof sketch: Consider the first action a_1 in c . Given that there are no PRECFALSE risks, all of a_1 's preconditions in D must be true in s_0 . Since there are no PRECOPEN risks, all of a_1 's preconditions in D' must be true in s_0 and thus a_1 will execute successfully in s_0 . A similar argument says that a_2 will execute successfully as long as no effect of action a_1 clobbered a precondition x of a_2 that was true in s_0 . However, if such a clobbering existed and x were in D then there would be a POSSCLOB(a_1, x) risk. If x were not in D then there would have to be an PRECOPEN(a_2) risk. Thus a_2 will execute successfully. By induction, the entire plan will execute successfully, including the last action which can encode the goal, as described above.

A plan with risks may also succeed when executed in the world because the actions are, in fact, completely modeled or the omissions in the model do not adversely effect the plan or because the plan succeeds with all possible action descriptions that are consistent with D . We can, however, define a subset of risks which are *critical*. A *critical* risk is not guaranteed to cause plan failure, but it does guarantee the possibility of failure.

In order to identify the critical risks, we first identify a set of *vulnerable* conditions in p . Intuitively, a condition is vulnerable if has only one source of support in a plan. Formally, we define vulnerability recursively as follows. A conjunct of the goal is vulnerable in a plan iff it is established exactly once by an action in the plan or the initial state s_0 . A precondition x of an action in a plan is vulnerable iff the action has an effect which establishes a vulnerable condition and x is established exactly once by either a prior action in the plan or by the initial state. The set of vulnerable conditions in a plan can be quickly computed in a simple pass backwards through the plan.

We can now define critical risks for the actions in a plan $p = a_1, \dots, a_n$. A PRECOPEN(a_i) risk is critical if a_i establishes a vulnerable condition. A HYPOTHEZIEDEFFECT(a_i, x) condition is critical if x is vulnerable. A POSSCLOB(a_i, x) is critical if x is vulnerable and a_i would execute successfully under the assumption that D is complete. A PRECFALSE(a_i, x) risk is critical if action

a_i establishes a vulnerable condition. The critical risks can also be easily computed in a single pass through the plan.

The following theorem states that critical risks guarantee possible plan failure.

Theorem: If $\text{FINDRISKS}(c, g, L)$ contains a critical risk, then there exists some D' such that $D \subset D'$, D' is consistent with the LCW in L , and $\neg \text{achieves}(D', s', g, c)$.

Proof sketch: We briefly consider each type of risk. In each case, we show there exists a D' such that a vulnerable condition of the plan will not be established or will be clobbered, which causes plan failure. An action a_i with a critical PRECOPEN risk can fail because there exists an x (or $\neg x$) that is a precondition of a_i in D' such that x will be false after a_{i-1} is executed. If a plan has a critical POSSCLOB(a_i, x) risk then it can fail if D' is identical to D except that action a_i has effect x , which will clobber a vulnerable condition. If $D' = D$, then a plan with a critical HYPOTHESEDEFFECT or PRECFALSE action will fail.

Plan selection algorithm

We now present our solution to the plan selection problem. Below we list a variety of policies for preferring one plan to another. Each policy takes two candidate plans, c_1 and c_2 , and returns true if c_1 is preferred to c_2 . We use $\text{risks}(c)$ as a shorthand for $\text{FINDRISKS}(c, g, L)$. We assume that a plan that achieves its goal with the given action descriptions is preferable to any plan that does not. Thus we assume that either both c_1 and c_2 achieve the goal given D or neither do.

- $RP_\emptyset(c_1, c_2) :: \text{risks}(c_1) = \emptyset$, and $\text{risks}(c_2)$ contains a critical risk.
- $RP_C(c_1, c_2) :: \text{risks}(c_1) \subset \text{risks}(c_2)$.
- $RP_<(c_1, c_2) ::$ the number of each type of risk in $\text{risks}(c_1)$ is less than that the number of that type of risk in $\text{risks}(c_2)$.
- $RP_w(c_1, c_2) :: \text{weighted}(c_1) < \text{weighted}(c_2)$ where $\text{weighted}(c)$ returns a real number by adding together the number of each type of risk multiplied by a pre-defined weight for that risk type.

Risks assessment can be incorporated with other methods for preferring plans. For example, most planners have an implicit preference for selecting the shortest plan that achieves the goal. These preference rules can either dominate the pre-existing preference method or be used only to break ties. Let $\text{TieBreak}(RP, c_1, c_2)$ return true if either the pre-existing preference method prefers c_1 to c_2 or it ranks them equal and $RP(c_1, c_2)$ is true.

The $\text{TieBreak}(RP_\emptyset, c_1, c_2)$ policy is the most conservative use of the techniques discussed in this paper. It only uses our techniques in the case where two candidate plans are considered equal (ignoring the incompleteness issues) and one has no risks and the other has at least one critical risk. We present it as an extreme case in which reasoning about incompleteness clearly (if infrequently) improves plan selection. As shown above, if the action descriptions in D are correct but incomplete then a plan with critical risks might fail, while a plan without risks cannot fail. If there

is no other basis for preferring one plan to another, it seems obviously beneficial to prefer the plan that cannot fail.

The other policies are more widely applicable but can prefer a plan that happens to fail to one that happens to succeed given D_{true} . This can happen with $RP_<$ and RP_w if plan c_1 has fewer risks than c_2 , but all risks in c_1 happen to correspond to real discrepancies between D and D_{true} while all the risks in c_2 happen not to.

Somewhat surprisingly, however, even if c_1 has a subset of the risks of c_2 , it is possible that c_1 will fail and c_2 will succeed. One reason for this is that a POSSCLOB(a_i, x) risk can be realized, i.e., a_i does have effect x in D_{true} , but then corrected by another action which has a $\neg x$ effect which is also missing from D . Further, it is possible that the risk will be corrected in c_2 but not in c_1 if only c_2 has the correcting action. If c_2 has some other risks which are not realized, then it will succeed while c_1 fails, even though c_1 has a subset of c_2 's risks.

While the policies other than $\text{TieBreak}(RP_\emptyset)$ can choose the inferior plan in certain cases, they are likely to improve plan selection, on average, under a reasonable set of assumptions about the distribution of planning problems. For example, for any given POSSCLOB risk, it seems much more likely that it will be realized than that it will be both realized and then corrected. If one plan has more POSSCLOB risks than another, then it is more likely to have a realized but uncorrected risk than the other, and therefore is more likely to fail.

Examples

We now describe three classes of planning problems for which it is beneficial to reason about the incompleteness of the action descriptions.

The first class of problems concerns the order of actions in a plan. Figure 2 shows a simple example designed to illustrate how our risk analysis can prefer one ordering of plan actions over another. In this figure, actions are drawn as links, from the action's preconditions on the left to the action's effects on the right.

Figure 2 shows a goal that can be achieved by executing two actions, a_1 and a_2 , in either order. If the action model is complete then both plans will achieve their goal.

Either plan can fail, however, if the model is incomplete. Candidate plan $C_1 = [a_1, a_2]$ could fail if a_2 has an effect $\neg r$ which clobbers a_1 's effect. Similarly, plan $C_2 = [a_2, a_1]$ could fail if a_1 clobbers a_2 's effect. However, C_2 could also

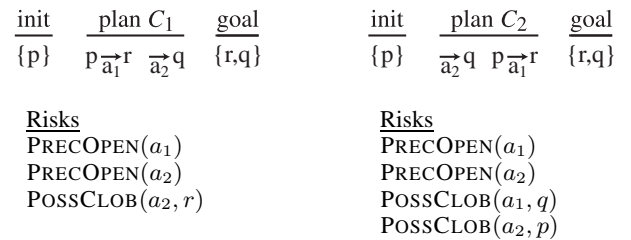


Figure 2: **Action order.** These two plans differ only in the order of the actions, but they have different numbers of risks.

init	plan C_1	goal
{p}	$\overset{\rightarrow}{a_1}q \quad q\overset{\rightarrow}{a_2}r \quad r\overset{\rightarrow}{a_3}s \quad s,p\overset{\rightarrow}{a_4}t$	{t}
<u>Risks</u>		
PRECOPEN(a_1)	POSSCLOB(a_1, p)	
PRECOPEN(a_2)	POSSCLOB(a_2, p)	
PRECOPEN(a_3)	POSSCLOB(a_3, p)	
PRECOPEN(a_4)		
init	plan C_2	goal
{p}	$\overset{\rightarrow}{a_1}q \quad q\overset{\rightarrow}{a_2}r \quad r\overset{\rightarrow}{a_3}s \quad \overset{\rightarrow}{a_5}p \quad s,p\overset{\rightarrow}{a_4}t$	{t}
<u>Risks</u>		
PRECOPEN(a_1)	POSSCLOB(a_5, s)	
PRECOPEN(a_2)		
PRECOPEN(a_3)		
PRECOPEN(a_4)		
PRECOPEN(a_5)		

Figure 3: **Additional steps.** The extra step a_5 in plan C_2 reduces the number of possible clobberings of action a_4 .

fail if a_2 has effect $\neg p$ which would clobber a_1 's known precondition. Thus, C_2 has more POSSCLOB risks than C_1 and both the $RP_{<}$ and RP_w policies would prefer C_1 to C_2 .

The preference for C_1 over C_2 is justified by imagining what we could add to D in order to create a D_{true} in which the plans would fail. For any combination of additional conditions and effects that would cause C_2 to fail, there is a corresponding modification that would cause C_1 to fail. However, in order to “match” the risk introduced by adding $\neg p$ as an effect of C_2 , we have to add both a precondition and an effect to C_1 . Thus, in the absence of any LCW information, C_1 seems the safer choice. On the other hand, if the given LCW eliminates the risks of plan C_2 , then it becomes the better plan to execute.

The second class of problems we consider are ones in which our plan selection policies will sometimes prefer plans with what seem like extra steps, i.e., steps that are not required to make the plan execute successfully given D . Figure 3 shows a simple example of how this could happen. If a precondition x of an action a_i is established by the initial state or by an early action then it has many chances to be clobbered. By adding in a “cleanup” step just before a_i to reestablish x , these potential clobberings risks are removed. However, the added step may also introduce risks, such as the PRECOPEN risk in Figure 3. If POSSCLOB risks are

init	plan C_1	goal	init	plan C_2	goal
{w}	$\overset{\rightarrow}{a_1}r,q \quad w,q\overset{\rightarrow}{a_2}s$	{r,s}	{w}	$w\overset{\rightarrow}{a_3}p \quad p\overset{\rightarrow}{a_4}r,s$	{r,s}
<u>Risks</u>			<u>Risks</u>		
PRECOPEN(a_1)			PRECOPEN(a_3)		
PRECOPEN(a_2)			PRECOPEN(a_4)		
POSSCLOB(a_1, w)					
POSSCLOB(a_2, r)					

Figure 4: **Operator choice.** The second plan is preferred because there are possible clobberings in the first.

weighted more than half the weight of PRECOPEN risks, then the RP_w will prefer plan C_2 in this example. Additionally, C_2 can be preferred to C_1 by any of the other selection policies if LCW excludes enough risks in C_2 .

The third and final class of problems arises when there are alternative actions with similar effects. The issues that arise in the ordering of actions and in adding additional steps also come up in the choice of actions to achieve the same goal or subgoal of a plan. The most obvious role of our techniques would be to prefer to use operators that are completely modeled over ones that are not. Additionally, Figure 4 shows two plans that look equally correct if the prospect of missing effects and preconditions is ignored, but one plan has more risks than the other.

Experiments

We conducted two experiments to measure how useful risk assessment is for plan selection. First, we implemented a modified version of the Fast Forward (FF) planning system (Hoffmann & Nebel, 2001) that exploits augmented domain descriptions to find the risks in each generated plan. The domain descriptions allow some preconditions and effects to be marked as “hidden”, i.e. knowledge that is in D_{true} but not in D . Thus, our version of FF can generate plans using D , and then evaluate them using D_{true} .

Our first attempt to evaluate our plan-selection framework was to slightly modify the domain descriptions for the planning domains that FF comes with. However, because the planning domains were highly crafted, we found that even small differences between D and D_{true} often meant that almost none of the plans generated using D would be successful in D_{true} . This did not seem to represent a realistic situation in which a domain model would be incomplete and yet still usable.

Each of our two experiments are based on a suite of pseudo-randomly generated triples $\langle g, D, D_{true} \rangle$, where g is a planning goal that can be solved in D but the solution may not achieve g when executed in D_{true} . There are precondition / effect dependencies in both D and D_{true} that reduce the number of plans produced by FF. Some additional dependencies are added to D_{true} but not to D so that between 65% and 85% of the solutions in D will achieve their goal in D_{true} . For these experiments, add effects in D_{true} were never hidden in D because if a hidden add-effect is needed to achieve goal g , then FF will not generate any successful plans when given D . Preconditions and delete effects were hidden at random. Also, each action was given, on average, the same number of preconditions to avoid introducing a bias.

In the first experiment, we measured the impact of risk assessment in action ordering decisions. The triples $\langle g, D, D_{true} \rangle$ were generated such that D (and D_{true}) contained exactly seven actions, all of which had to be executed exactly once to achieve g . Our risk-assessment techniques were used to select which ordering of these seven actions should be executed.

In the second experiment, we measured the impact of risk assessment in operator selection decisions. The triples $\langle g, D, D_{true} \rangle$ were generated so that g is always achieved

Run	Percentiles of the number of risks distribution										Avg # of risks	Dev. of risks	Min # of risks	Max # of risks	Number of plans
	t_{10}	t_{20}	t_{30}	t_{40}	t_{50}	t_{60}	t_{70}	t_{80}	t_{90}	t_{100}					
AO	87.6	86.2	84.7	84.0	83.1	82.3	81.4	80.3	78.9	76.7	49.4	4.3	42.0	74.0	195254
OC	85.4	83.0	81.4	80.9	79.8	79.0	77.9	77.1	76.3	74.7	72.0	7.8	55.0	115.0	341469

Table 1: Impact of risk assessment on likelihood of successfully executing plans.

by a sequence of length 10. For each position i in the plan, there are two possible choices $a_{i,0}$ and $a_{i,1}$. These two actions share a common effect and a common precondition (which is the common effect of both $a_{(i-1),0}$ and $a_{(i-1),1}$). Our risk-assessment techniques were used to decide which combination of operators should be executed.

Table 1 gives statistics showing how risks and success percentages are related for the two experiments. The first row presents results for the action ordering (abbreviated “AO”) and the second presents results for operator choice (“OC”). Within each row, there are 10 columns that show the likelihood of successfully executing a plan drawn at random from different subsets of the set of generated plans. The final five columns show general statistics about the distribution of risks in the set of generated plans.

Within each row, the success percentages in the columns labelled t_{10}, \dots, t_{100} are derived from different subsets of the set of generated plans. For t_k , the subset contains all plans whose number of risks are in the lowest k th percentile of distribution. For example, t_{50} includes all plans with fewer risks than the median number of risks, and t_{100} includes all plans.

The results show what a significant impact risk assessment can have. For both runs, generating two plans and preferring the one with fewer risks will, on average, increase the chance of success for roughly 75% to 80%. Generating more candidates continues to provide benefits, as selecting a plan from t_{10} increase the likelihood of success to over 85%.

Related Work

Much previous work has addressed the problem of planning with incomplete state information and non-deterministic or conditional effects (e.g, Kushmerick, Hanks, & Weld (1995); Smith & Weld (1998)). This is similar to the problem of planning with incomplete action models in the sense that both problems are concerned with uncertainty about the effects of actions. One important difference, however, is that non-deterministic planning systems demand even more elaborate action descriptions than traditional planners. For example, the action descriptions are required to describe all the different possible sets of effects that an action might have. In contrast, our techniques can improve planning even without any additional information, and we provide a framework in which any additional effort to produce LCW statements can be factored into the planning process. Further, our techniques are designed to exploit various types of information, even statements about which fluents an action does *not* effect.

Additionally, the objective of work on non-deterministic planning is usually to generate plans that are guaranteed to succeed, or are guaranteed to succeed with some probability.

As a result, even assessing a probabilistic or conditional plan to determine if it will succeed requires exponential computation with the length of the plan. In contrast, our methods simply prefer to execute plans with fewer risks. Further, our techniques are linear in the length of the plan and the size of the state, though we do require the planner to generate multiple plans to choose from.

Prior work has addressed the complementary problem of improving action models between planning episodes. One approach has been to develop knowledge acquisition systems that help domain experts convey their knowledge to a computer. For example, Tecuci *et al.* (Tecuci *et al.*, 1999) present techniques for producing hierarchical if-then task reduction rules by demonstration and discussion from a human expert. A second approach is to develop techniques for improving action descriptions based on the observed results of executing actions (Gil, 1994; Wang, 1995; Oates & Cohen, 1996). Our techniques are complementary since our methods are designed to improve planning when there are incomplete action descriptions. One possible synergy between these two lines of research would be to develop techniques for automatically learning LCW or helping to elicit it from the domain experts.

Conclusions and Future Work

This work is motivated by our belief that the primary obstacle to the practical application of planning technology is the difficulty of obtaining task models. This problem should be addressed both by developing better techniques for eliciting models, and by adapting planning algorithms to use models that are less difficult to elicit. In this paper, we pursue the latter approach.

Our future work includes incorporating other types of knowledge about how to do things into a planning process that reasons about incomplete action models. As mentioned above, if an important effect or precondition is missing from the action descriptions given to a planner, it can have little chance of achieving its goal. Indeed, once we forgo the completeness assumption, the set of valid plans can no longer be deduced from the action descriptions. Thus, it becomes especially useful for human experts to provide additional information about how to accomplish goals and subgoals by, for example, providing a library of pre-defined goal-decomposition rules (also called recipes and hierarchical task networks). We are particularly interested in using LCW information to allow more flexible use of goal-decomposition knowledge.

Another line of future research is to integrate the techniques presented here into a system that integrates (re)planning and execution. The planning system will keep track of what actions it has executed and what state information it has sensed at different time points. Our techniques

for assessing risks can be applied to the execution history in order to determine which fluents it should most rely on. If the preconditions of an executed action had many risks associated with it, then the planner should be wary of using that action's presumed effects to support the preconditions of future actions.

Acknowledgements

We gratefully thank Charles Rich, Candace Sidner, and the anonymous reviewers for their insightful comments.

References

- Babaian, T., and Schmolze, J. G. 2000. PSIPlan: Open World Planning with ψ -forms. In *Proc. 5th Int. Conf. on AI Planning Systems*, 292–307.
- Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 115–125.
- Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *Eleventh Intl Conf on Machine Learning*, 87–95.
- Golden, K. 1998. Leap before you Look: Information Gathering in the PUCCINI planner. In *Proc. 4th Int. Conf. on AI Planning Systems*, 70–77.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76:239–286.
- Levesque, H. J. 1996. What is planning in the presence of sensing. In *Proc. 13th Nat. Conf. AI*, 1139–1146.
- McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of Artificial Intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502.
- Oates, T., and Cohen, P. 1996. Searching for planning operators with context-dependent and probabilistic effects. In *Proc. 13th Nat. Conf. AI*, 863–868.
- Reiter, R. 1991. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press. 359–380.
- Smith, D., and Weld, D. 1998. Conformant Graphplan. In *Proc. 15th Nat. Conf. AI*, 889–896.
- Tecuci, G.; Boicu, M.; Wright, K.; Lee, S.; Marcu, D.; and Bowman, M. 1999. An integrated shell and methodology for rapid development of knowledge-based agents. In *Proc. 16th Nat. Conf. AI*, 250–257.
- Wang, X. 1995. Learning by observation and practice: an incremental approach for planning operator acquisition. In *Proc. 12th Int. Conf. on Machine Learning*, 549–557.