To appear in *Proc. 15th National Conf. on AI*, 1998.

Conformant Graphplan

David E. Smith

NASA Ames Research Center
Mail stop 269–2
Moffett Field, CA 94035
de2smith@ptolemy.arc.nasa.gov

Daniel S. Weld

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
weld@cs.washington.edu

Abstract

Planning under uncertainty is a difficult task. If sensory information is available, it is possible to do *contingency planning* – that is, develop plans where certain branches are executed conditionally, based on the outcome of sensory actions. However, even without sensory information, it is often possible to develop useful plans that succeed no matter which of the allowed states the world is actually in. We refer to this type of planning as *conformant planning*.

Few conformant planners have been built, partly because conformant planning requires the ability to reason about disjunction. In this paper we describe Conformant Graphplan (CGP), a Graphplan-based planner that develops sound (non-contingent) plans when faced with uncertainty in the initial conditions and in the outcome of actions. The basic idea is to develop separate plan graphs for each possible world. This requires some subtle changes to both the graph expansion and solution extraction phases of Graphplan. In particular, the solution extraction phase must consider the unexpected side effects of actions in other possible worlds, and must confront any undesirable effects.

We show that CGP performs significantly better than two previous (probabilistic) conformant planners.

Introduction

There are two basic approaches to planning when uncertainty is present in the world:

1. *Contingency planning* – develop plans where some branches are executed conditionally, based on the outcome of sensory actions.
2. *Conformant planning* – develop non-conditional plans that do not rely on sensory information, but still succeed no matter which of the allowed states the world is actually in.

When sensory actions are cheap, but effectory actions are expensive or dangerous, contingency planning makes a great deal of sense. However, both of these options have their place. In situations where sensing is difficult or impossible, conformant planning may be the best (or only) alternative. As an example, it might be difficult to determine if a particular work surface is clean, or if an instrument is sterile. Yet it may be a simple matter to clean the work surface or sterilize the instrument, thus resolving the uncertainty.

In these two examples, conformant planning involves removing some of the uncertainty in the world by forcing propositions into known states. More generally, conformant planning involves case analysis to make sure the goals are achieved in any of several situations. For example, suppose a patient has one of several diseases, but it cannot be determined precisely which one. Finding a drug therapy plan that covers all the cases (without bad drug interactions) is a conformant planning problem.

There are a number of planning systems that do contingency planning, including: Warplan-C (Warren 1976), CNLP (Peot and Smith 1992), SENSP (Etzioni, et al. 1992), Plinth (Goldman and Boddy 1994), XII (Golden, Etzioni and Weld 1994), Cassandra (Pryor and Collins 1996), C-Buridan (Draper, Hanks and Weld 1994), and DTPOP (Peot 1998). However, only Buridan (Kushmerick, Hanks and Weld 1995), C-Buridan and UDTPOP (Peot 1998) do conformant planning. All three of these systems are probabilistic planners and are limited to propositional actions. Furthermore, these planners are incredibly slow – so slow that they make planners like UCPOP (Penberthy and Weld 1992) and Prodigy (Veloso et al. 1995) look positively zippy.

Recently, there has been a great deal of interest in Graphplan (Blum and Furst 1995, 1997), which seems to have a significant performance advantage over planners like UCPOP and Prodigy. As originally described, Graphplan is limited to simple STRIPS operators. In this paper we describe Conformant Graphplan (CGP) – a modification of Graphplan that deals with uncertainty by constructing conformant plans. Unlike Buridan, C-Buridan and UDTPOP, CGP is not probabilistic – the initial conditions may contain disjunction and actions may have uncertain outcomes, but the system has no notion of the likelihood of different propositions.

The basic idea in CGP is to create a different plan graph for each possible world. Unfortunately, this doesn't quite work, because, in the absence of sensing actions, an action cannot be confined to one possible world. This complicates the solution extraction phase of Graphplan. In particular, when CGP selects an action in one possible world, it must consider what might happen if it is in another possible world. If the consequences of the action are undesirable in that other possible world, CGP must *confront* the action in that other possible world. In some cases it is possible to recognize problematic interactions between worlds and derive mutual exclusion relationships between propositions in the different possible worlds. This results in a considerable effi-

ciency improvement but complicates the mutual exclusion rules for CGP.

In the next section we briefly review the Graphplan algorithm. In Section 3 we describe the modifications necessary to make this algorithm handle uncertainty in the initial conditions. In Section 4 we extend the algorithm to handle actions with uncertain outcomes. Finally, we show that CGP dramatically outperforms previous conformant planners.

Graphplan background

Graphplan (Blum and Furst 1995, 1997) accepts action schemata in the STRIPS representation – preconditions are conjunctions of positive literals and effects are a conjunction of positive or negative literals (i.e., composing the add and delete lists). Graphplan alternates between two phases: *graph expansion* and *solution extraction*. The graph expansion phase extends a *planning graph* until it has achieved a necessary (but insufficient) condition for plan existence. The solution extraction phase performs a backward-chaining search for an actual solution; if no solution is found, the cycle repeats.

The planning graph contains two types of nodes, proposition nodes and action nodes, arranged into levels. Even numbered levels contain proposition nodes, and the zeroth level consists precisely of the propositions that are true in the initial state of the planning problem. Nodes in odd-numbered levels correspond to action instances; there is an odd-numbered node for each action instance whose preconditions are present (and are mutually consistent) at the previous level. Directed edges connect proposition nodes to the action instances (at the next level) whose preconditions mention those propositions. Directed edges connect action nodes to subsequent propositions made true by the action's effects.

The most interesting aspect of Graphplan is its use of local consistency methods during graph creation – this appears to yield a dramatic speedup during solution extraction. Graphplan defines a binary mutual exclusion relation ("mutex") between nodes in the same level as follows:

1. Two action instances at level i are mutex if either
   - *Interference / Inconsistent Effects*: one action deletes a precondition or effect of another, or
   - *Competing needs*: the actions have preconditions that are mutually exclusive at level i-1.
2. Two propositions at level i are mutex if all ways of achieving the propositions (i.e. actions at level i-1) are mutex.

Suppose that Graphplan is trying to generate a plan for a goal with n conjuncts, and it has finally extended the planning graph to an even level, i, in which all goal propositions are present and none are pairwise mutex. Graphplan now searches for a solution plan by considering each of the n goals in turn. For each such proposition at level i, Graphplan chooses an action a at level i-1 that achieves the goal. This is a backtracking choice – all possible actions must be considered to guarantee completeness. If a is consistent (non-mutex) with all actions that have been chosen so far at this level, then Graphplan proceeds to the next goal, otherwise if no such choice is available, Graphplan backtracks. After Graphplan has found a consistent set of actions at level i-1 it recursively tries to find a plan for the set of all the preconditions of those actions at level i-2. The base case for the recursion is level zero – if the propositions are present there, then Graphplan has found a solution. If, on the other hand, Graphplan fails to find a consistent set of actions at some level and backtracking is unsuccessful, then it continues to alternate between growing the planning graph and searching for a solution (until it reaches a set limit or the graph levels off).

Negated Preconditions

Although methods for handling negated preconditions were not presented in (Blum and Furst 1995, 1997), they are both straightforward and essential prerequisites for handling conditional effects. Clearly proposition p and $\neg p$ are mutually exclusive in any given level. Whenever an action instance deletes a proposition (i.e. has a negated literal as an effect), one must add that negative literal to the subsequent proposition level in the planning graph.

Conditional effects

As originally described, Graphplan supports only simple STRIPS operators. Recently, several authors have described methods that allow Graphplan to handle operators with conditional effects (Gazen and Knoblock 1997, Koehler et al. 1997, Anderson, Smith and Weld 1998). For pedagogical reasons we adopt the simple approach used by Gazen and Knoblock, which breaks operators with conditional effects up into a number of separate operators (by considering all minimal consistent combinations of antecedents in the conditional effects). To illustrate, consider the ADL operator:

OP	pre:	P
	eff:	(and E (when C_1 F) (when C_2 G))

This operator would be expanded into the following four STRIPS operators:

OP1	pre:	(and P $\neg C_1$ $\neg C_2$)
	eff:	E
OP2	pre:	(and P $\neg C_1$ C_2)
	eff:	(and E G)
OP3	pre:	(and P C_1 $\neg C_2$)
	eff:	(and E F)
OP4	pre:	(and P C_1 C_2)
	eff:	(and E F G)

We will refer to these four operators as *aspects* of the original ADL operator.

Conformant Graphplan

To begin, we limit consideration to cases where uncertainty occurs only in the initial conditions (i.e. action effects are certain), but we relax this restriction in the following sec-

tion. With Graphplan, the initial conditions are specified as a conjunction of positive and negative literals. We augment this language to allow the use of disjunction (or) and exclusive or (xor).

Plan graph expansion

The basic idea behind conformant Graphplan is to express the uncertainty in the initial conditions as a set of completely specified *possible worlds*, and run the Graphplan expansion procedure on each of these possible worlds in parallel. To do this, we first initialize a separate plan graph for each possible world. The expansion phase of Graphplan is then the same as for a normal plan graph; we just have to do it for each of the possible worlds.

To illustrate, consider the simple "bomb in the toilet" problem from (Mcdermott 1987). There are two packages, one of which contains a bomb. Dunking the package with the bomb in it renders the bomb disarmed. We formalize the initial conditions as (and armed (xor In(P1) In(P2))). The goal is ¬Armed and the single operator

```
Dunk (?pkg)      pre:
                 eff:    (when In(?pkg) ¬Armed)
```

has only one aspect with non-empty effects:

```
Dunk* (?pkg)     pre:    In(?pkg)
                 eff:    ¬Armed
```

Figure 1 shows the first level of the possible world plan graphs (PWPGs) for this example. In world w_1, Dunk*(P1) can be used to achieve ¬Armed, while in world w_2, Dunk*(P2) can be used to achieve ¬Armed. In each of the two graphs there is a mutex relationship between the Dunk* action and the persistence of Armed, because a proposition and its negation cannot both be true. Note that the two plan graphs in Figure 1 are separate – there are no links that cross from one world to the other. In the solution extraction phase, we have to consider interactions between aspects in different worlds but the graphs will remain separate. Later on, we introduce explicit mutual exclusion relationships between propositions in different PWPGs.

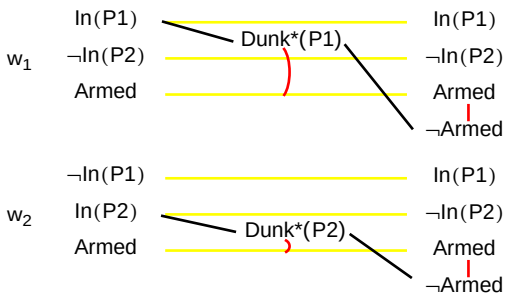

Figure 1: Possible world plan graphs for the basic bomb in toilet problem. Gray arcs indicate no-op steps for persistence of propositions from one level to the next. Arcs between steps and arcs between propositions are mutex relationships.

For convenience we use the notation p:w to refer to the proposition p in world w. Similarly, we use a:w to refer to aspect a in world w.

Solution extraction

Graphplan searches backwards for a plan whenever all the goal literals occur at the current level of the plan graph, and none of those literals are mutex with each other. With uncertainty present, we need to guarantee that the goal is satisfied in all possible worlds. This requires several changes to the solution extraction phase of Graphplan. First, CGP searches for a plan at an even level only when the goal literals appear at that level in each of the possible world plan graphs (and none of those literals are mutex with each other). Second, the search proceeds backwards one level at a time, simultaneously through all of the possible world plan graphs[1]. Third, as is shown later, the search needs to consider interactions across different possible worlds.

Consider the possible world plan graphs in Figure 1 and recall that the goal is ¬Armed. At level 1, ¬Armed is present in both PWPGs, so CGP should search backwards for a plan. The goal ¬Armed in world 1 can only be achieved using the aspect Dunk*(P1):w_1. Likewise, ¬Armed in world 2 can only be achieved using the aspect Dunk*(P2):w_2. These two aspects are not mutex, and their subgoals at level 0 are achieved by the initial conditions. Thus, it appears that the plan of dunking both packages achieves the goal (and in this case it actually works to dunk both packages).

Unfortunately, there is a problem with this simple-minded approach. If we choose to perform an action, we must consider its effects in all worlds, not just the specific world in which the action was added. This is because (without sensing) the planner doesn't know which of the worlds it is really in and therefore cannot confine the effects of an action to just one possible world. To compound matters, each world is different, so performing an action may lead to one aspect of the action in one possible world and a completely different aspect in another possible world.

To illustrate this problem, we augment the bomb in the toilet problem by adding both a precondition (that the toilet is not clogged) and an unconditional effect (that the toilet is clogged afterwards) to the Dunk operation:

```
Dunk (?pkg)      pre:    ¬Clogged
                 eff:    (and Clogged
                              (when In(?pkg) ¬Armed)
```

This action has two aspects:

```
Dunk– (?pkg)     pre:    (and ¬Clogged ¬In(?pkg))
                 eff:    Clogged

Dunk+ (?pkg)     pre:    (and ¬Clogged In(?pkg))
                 eff:    (and Clogged ¬Armed)
```

We also augment the initial conditions to include the fact that the toilet is not initially clogged. With these additions the problem cannot be solved for all possible worlds, because we will not be able to dunk both packages. Figure 2 shows the first level of the plan graph for this new problem.

1. This is not strictly necessary. Other search strategies (like searching the worlds sequentially) could also be used, but interactions between aspects in different possible worlds would not be recognized as early.

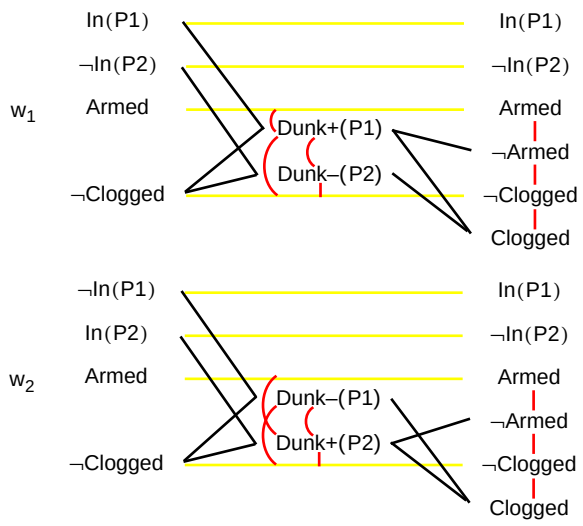

Figure 2: A Possible worlds plan graph when the dunk operation clogs the toilet.

At level 1, ¬Armed is present in both PWPGs, so CGP would search backwards for a plan. The goal ¬Armed:w_1 can only be achieved using the aspect Dunk+(P1):w_1. Likewise, the goal ¬Armed:w_2 can only be achieved using the aspect Dunk+(P2):w_2.

Next we must consider the consequences of these two aspects in the opposite possible world. Consider the aspect Dunk+(P1):w_1. To guarantee that aspect in world 1, we must perform the action Dunk(P1), which results in the aspect Dunk+(P1):w_1 as well as the aspect Dunk–(P1):w_2. Since Dunk–(P1):w_2 is mutex with Dunk+(P2):w_2, the plan of dunking both packages fails. We formalize these intuitions as:

Definition 1: Suppose aspect a:w is present at level i in a PWPG and a':v ($v \neq w$) is another aspect of the *same* action at level i in the PWPG. We say that a:w *possibly induces* a':v, and vice versa.

In the example above, Dunk+(P1):w_1 possibly induces Dunk–(P1):w_2 and vice versa. Likewise for Dunk–(P2):w_1 and Dunk+(P2):w_2.

Sometimes it is possible to prevent an undesirable possibly induced aspect using *confrontation*[2]. To prevent or confront aspect a':v, the planner needs to assure that its precondition is false at level i-1 and remains false during the execution of the other aspects at level i. (Otherwise, a':v might fire if a is executed after its precondition becomes established.) Thus if a':v has preconditions p_1, p_2, ..., p_n the planner must find a precondition p_i:v that can be made false at level i-1 and can be held false until level i+1. One easy way to implement this is to check that the no-op operation persisting ¬p_j:v

1. exists at level i, and
2. is not mutex with any other aspect in the plan at level i.

If so, this no-op is added to the plan, and its precondition ¬p_j:v is added to the subgoals at level i-1. Note that when doing confrontation, there may be a choice of which precondition p_j:v to confront. This provides an additional backtrack point during the solution extraction process.

With the notions of possibly induced aspects and confrontation, we can now describe the solution extraction phase of CGP more precisely. Let i+1 be the newest proposition level in the graph and let S_{i+1} (the set of subgoals to be achieved at level i+1) be initialized to the set of goals G tagged with each possible world: $\{g{:}w \mid g \in G, w \in W\}$.

1. Initialize A_i (the set of aspect-world pairs chosen at level i) to the empty set.
2. For each subgoal $g{:}w \in S_{i+1}$ choose (backtrack point) an aspect a:w at level i that achieves g:w, and is consistent (not mutex) with every other aspect in A_i. Unless a:w is already in A_i:
   - Add a:w to the set A_i.
   - For each precondition p:w of a:w, unless p:w is already in S_{i-1}, add p:w to S_{i-1}.
3. Let D_i refer to the set of all aspects possibly induced by aspects in A_i. For each aspect a:w in A_i, confront (backtrack point) each possibly induced aspect in D_i that is mutex with a:w.
4. If i=1, return the completed plan A_1, A_3, ..., A_{n-1}, otherwise reduce i by 2 and repeat.

Note that in doing the confrontation in step 3, a new no-op may get added to the set A_i. This no-op must also be made safe from all aspects in D_i. As a result, step 3 continues as long as A_i keeps growing. (D_i does not grow during this process because no-ops don't induce other aspects.)

Induced mutex

In the example in Figure 2, we had to commit to the aspect Dunk+(P1):w_1 during solution extraction before we discovered that the possibly induced aspect Dunk–(P1):w_2 could not be prevented. Since Dunk–(P1):w_2 was mutex with the desired aspect Dunk+(P2):w_2, the plan failed. In fact, we can discover this problem much earlier. The key is to recognize that Dunk+(P1):w_1 always induces the aspect Dunk–(P1):w_2 at that level. Since Dunk–(P1):w_2 is mutex with Dunk+(P2):w_2 this means that Dunk+(P1):w_1 should also be labelled as mutex with Dunk+(P2):w_2. We formalize this notion with the following definition:

Definition 2: Suppose aspect a:w is present at level i of a PWPG, and a':v ($v \neq w$) is another aspect of the *same* action at level i in the PWPG. We say that a:w *necessarily induces* a':v at level i if for every precondition p_j of a':v either:

¬p_j:v is not present at level i-1, or

¬p_j:v is mutex with some precondition of a:w at level i-1.

2. In UCPOP, confrontation is used to prevent undesirable conditional effects of a chosen action within a single world (no uncertainty). Here we do not need to do this within a given possible world because we expanded conditional effects out into aspects that are mutually exclusive. However, even with this expansion into aspects, we still need to prevent possibly induced aspects from occurring in other possible worlds.

These conditions guarantee that a':v cannot be confronted at level i.

In the example above, we note that Dunk–(P1):w_2 cannot be confronted at level 1, because none of the negations of its preconditions are present at level 0. As a result, Dunk+(P1):w_1 necessarily induces Dunk–(P1):w_2. (The converse also holds in this case, as do similar relationships for the aspects of Dunk(P2).)

Given the definition for necessarily induces, we can now state the induced mutex rule:

> **Induced mutex:** If aspect a:w necessarily induces a':v at level i and a':v is mutex with another aspect b:u at level i, then a:w is also mutex with b:u.

This rule is illustrated in Figure 3. Figure 4 illustrates just a few of the resulting induced mutex relationships for the example of Figure 2. The most important of these is that Dunk+(P1):w_1 and Dunk+(P2):w_2 are mutex. This means that, according to the usual mutex rules, ¬Armed:w_1 and ¬Armed:w_2 are mutex. As a result, no backward search will take place for this problem at level 2.

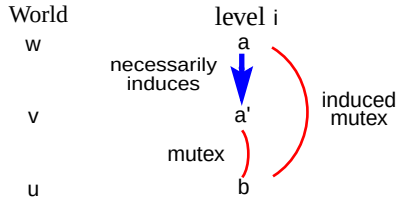

Figure 3: Induced mutex relationships.

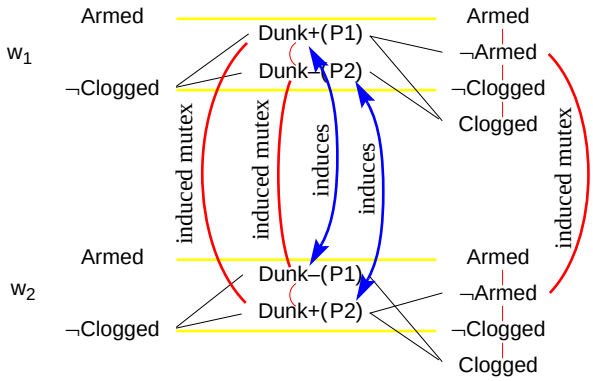

Figure 4: Two of the induced mutex relationships for the bomb in the toilet problem. For clarity, the static preconditions In(P1) and In(P2), and some of the other mutex relationships have been omitted from the PWPG. For this problem, there are many more induced mutex relationships between the dunk operations and persistence actions. For example, Dunk+(P1):w_1 is mutex with the persistence of Armed in w_1, so Dunk–(P1):w_2 will be mutex with the persistence of Armed in w_1.

Note that deriving induced mutex relationships is strictly an efficiency measure. Without it CGP still discovers conflicts due to induced aspects by virtue of the confrontation mechanism (during solution extraction). However, as we will illustrate in the results section, deriving induced mutex relationships substantially improves the performance of CGP.

Conversely, deriving induced mutex relationships, does not eliminate the need for confrontation during solution extraction. Induced mutex relationships only eliminate conflicts do to necessarily induced aspects. They do not eliminate potential conflicts due to possibly induced aspects.

Actions with uncertain outcomes

So far we have assumed that all uncertainty results from uncertainty in the initial conditions. We now expand our treatment to include actions with uncertain outcomes. In keeping with the representation used in Buridan (Kushmerick, Hanks and Weld 1995), we specify uncertain outcomes for an action by replacing the effects by a list of disjoint outcomes, each element of which is a traditional effects list. (An ordinary STRIPS action would therefore have only a single element in its outcomes list.)

For example, suppose that after performing a Dunk operation it is uncertain whether the toilet will be clogged or not. Thus, the Dunk operator becomes:

Dunk (?pkg)	pre:	¬Clogged
	outcomes:	(when In(?pkg) ¬Armed)
		(and Clogged
		(when In(?pkg) ¬Armed))

As before, we break this action into two aspects:

Dunk– (?pkg)	pre:	(and ¬Clogged ¬In(?pkg))
	outcomes:	Ø
		Clogged
Dunk+ (?pkg)	pre:	(and ¬Clogged In(?pkg))
	outcomes:	¬Armed
		(and Clogged ¬Armed)

where Ø signifies the empty effect. (Note that we need to include the empty outcome for aspect Dunk–, to indicate that the aspect may have no effect.)

Plan graph expansion

When uncertainty is confined to the initial conditions, expansion of each PWPG was essentially the same as for an ordinary plan graph. For actions with uncertain outcomes, this is no longer the case – each time such an action occurs in the plan graph, the number of possible worlds is multiplied by the number of possible outcomes of the action. Consider the aspect Dunk+(P1) in the plan graph shown in Figure 5. The

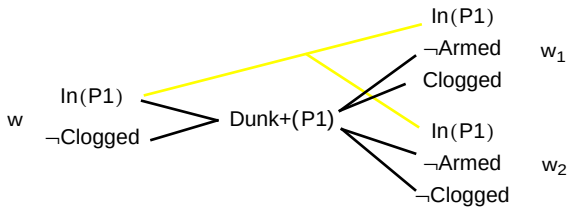

Figure 5: World splitting from an uncertain outcome.

world w is split into two possibilities, w_1 and w_2 correspond-

ing to the two outcomes of Dunk+(P1). When a world is split, the outcomes of other actions must be split as well. For example, the persistence action for In(P1) supports the proposition In(P1) in both of the new possible worlds.

In our example, suppose that the aspect Dunk–(P2) also occurs at this level of the plan graph. It too has an uncertain outcome, so world w really must be split into four possibilities, as shown in Figure 6.

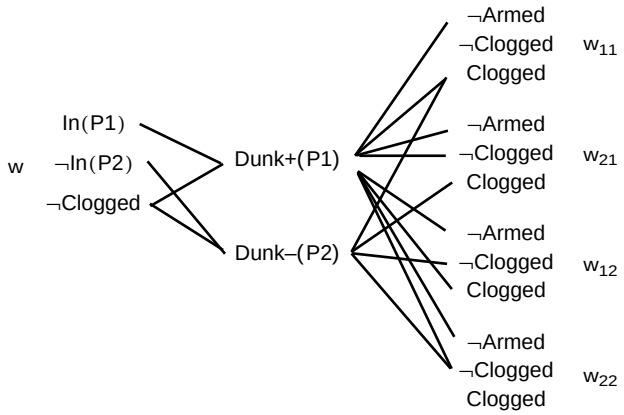

Figure 6: World splitting from two uncertain outcomes.

The splitting operation requires changes to the expansion phase of CGP. As before we add each of the new aspects to the plan graph for a possible world. However, we delay adding the propositions to the next proposition level until all the aspects are added (so that we know how many ways the possible world must be split.)

Consider one particular world w at the current level of the plan graph. Let $U=\{u_1,\ldots, u_n\}$ be the aspects in that world with uncertain outcomes and let o_j be the number of different outcomes for aspect u_j. World w will be split into

$$\prod_{j=1}^{n} o_j$$

new worlds. We label these new possible worlds with a sequence of subscripts, $s_1 \ldots s_n$, one subscript for each of the n aspects with uncertain outcomes. The total set of possible worlds derived from w will be:

$$\left\{ w_{s_1, \ldots, s_n} \mid s_j = 1, \ldots, o_j \right\}$$

If a is an aspect with no uncertainty, then its effects get added to each of the above possible worlds, and links are added from a to those effects. However, if a is an uncertain aspect u_m, the effects of outcome k of a will only get added to the subset of the worlds corresponding to outcome k, namely, the subset where $s_m=k$:

$$\left\{ w_{s_1, \ldots, s_{m-1}, k, s_{m+1}, \ldots, s_n} \mid s_j = 1, \ldots, o_j, j \neq m \right\}$$

As before, these effects get linked to the aspect a.

If aspects with uncertain outcomes appear frequently in the plan graph, this can lead to an explosion in the number of possible worlds. One trick for cutting down the number of possible worlds is to take advantage of mutex relationships between aspects. In particular, if two mutex aspects have uncertain outcomes, they can share possible worlds, so the number of possible worlds required is the maximum (rather than the product) of the number of outcomes for the two aspects. In general, sharing worlds between three or more aspects requires that they all be pairwise mutually exclusive. This leads to the interesting side problem of partitioning the uncertain aspects into mutex cliques in order to minimize the total number of worlds created.

Mutual exclusion

Remarkably enough, world splitting does not require any changes to the solution extraction algorithm we gave in the previous section. However, actions with uncertain outcomes do mandate a slight change in the definition of mutex relationships. Since we cannot predict which outcome of an uncertain aspect will actually occur, we must be conservative and allow for the worst. In particular, if *any* outcome of an aspect deletes a precondition or *possible* effect of another aspect, the two aspects are mutex. To fix this, we modify one of Graphplan's basic mutex rules as follows:

- *Interference / Inconsistent Effects*: Two aspects at level i are mutex if one action can *possibly* delete a precondition or *possible* effect of another.

Results

CGP is implemented in Common Lisp and accepts domains in the PDDL format, augmented to allow disjunction in the initial conditions and operators with uncertain outcomes. The implementation generates a single plan graph in which each proposition at each level is labelled with the possible world that it is in.

Table 1 shows the performance of Buridan (Kushmerick, Hanks and Weld 1995), UDTPOP (Peot 1998), CGP, and CGPx (CGP without induced mutex relationships) on a series of increasingly complicated "bomb in the toilet" problems. The domain contains two operator schemas, Dunk and Flush. Dunk clogs the toilet unconditionally and disarms the bomb if it is present in the dunked package. Flush clears a clogged toilet. The problems vary the number of toilets available (from 1 to 3) and the number of packages that the bomb can be in (from 2 to 6).

Both Buridan and UDTPOP are propositional, so we created propositional versions of the domains and problems for these systems. These planners are also probabilistic; so for the initial conditions, we divided the probability mass equally among each one of the initial possible worlds. As with CGP, the operators for these test problems all had certain (probability 1) outcomes.

As can be seen in the table, Buridan's performance was abysmal; it was only able to solve the three tiniest (two package) problems. On all other problems, progress was halted after the generation of 100,000 nodes (which corresponded to somewhere between 5 and 7 minutes of CPU time). UDT-

Toilets	Pkgs	Steps	Buridan	UDTPOP	CGPx	CGP
1	2	3	.44	.072	.010	.009
	3	5	*	.70	.073	.021
	4	7	*	8.4	3.24	.057
	5	9	*	149	119	.21
	6	11	*	*	*	.94
2	2	2	.57	.15	.007	.006
	3	4	*	3.0	.024	.022
	4	6	*	99	.071	.04
	5	8	*	*	156	1.2
	6	10	*	*	*	2.7
3	2	2	1.0	.26	.008	.007
	3	3	*	5.2	.011	.011
	4	5	*	*	.15	.070
	5	7	*	*	.92	.15
	6	9	*	*	6.4	.27

Table 1: Mean CPU times for various instances of the bomb clogs toilet example. All times are in seconds on a Macintosh 8600/300 using MCL 4.2. A * indicates that no solution was found in 5 CPU minutes. All tests were run 10 times. The largest variation (σ = 7%) was on the smallest (first) problem.

POP performed much better, but still had difficulty with larger numbers of packages.

CGPx (CGP without induced mutex) outperformed UDTPOP slightly on the single toilet problems. However, as the number of toilets was increased, the gulf widened considerably. In fact, for three toilets and four or more packages CGPx is more than two orders of magnitude faster than UDTPOP. We speculate that this advantage is due to Graphplan's ability to deal with actions in parallel. For example, in the three toilet problems, three dunk operations can be done in parallel at level 1, three flush operations can be done in parallel at level three, and so on. As a result, the depth of search required to find a plan is significantly lower than for more traditional partial-order planners.

Although CGPx performs better than UDTPOP, full CGP with induced mutex relationships is far superior. In fact CGP is one to two orders of magnitude faster than CGPx on the hardest problems. CGP is two to three orders of magnitude faster than UDTPOP on the harder problems.

Despite this impressive improvement, even CGP slows down significantly as the number of packages grows beyond seven. The reason is that, although this problem is conceptually trivial, it is combinatorially quite hard. As the number of packages grows, the solutions consist of progressively longer sequences of dunking and flushing operations. For example, with one toilet and seven packages, 13 steps are required. The trouble is, the packages can be dunked in any of 7! possible orders. So CGP spends all of its time generating and investigating these different possible orderings.

One obvious concern is how CGP performs as the amount of uncertainty increases. If there are k independent uncertain propositions in the initial conditions, there will be 2^k possible worlds. To see what effect this has on CGP, we added between one and five irrelevant uncertain propositions to the initial conditions for the single toilet problems. The results are shown in Table 2. The additional uncertainty significantly degrades performance, more so for the problems that require a deeper plan graph. It is certainly possible to use preprocessing to eliminate irrelevant propositions, and hence control the growth in number of possible worlds, but this will not be enough to allow CGP to solve real problems with many real sources of uncertainty.

Pkgs	Irrelevant uncertain propositions					
	0	1	2	3	4	5
2	.009	.018	.065	.46	3.4	29
3	.021	.070	.53	13.3	*	*
4	.057	.33	9.1	*	*	*
5	.21	3.2	*	*	*	*
6	.94	38	*	*	*	*

Table 2: Mean CPU times for varying number of packages (one toilet) with additional irrelevant uncertain propositions in the initial conditions. Note that five additional irrelevant propositions corresponds to multiplying the number of possible worlds by 2^5.

Conclusions

CGP is a Graphplan-based planner that does conformant planning; it attempts to construct non-contingent plans that succeed no matter which of the allowed states the world is actually in. The central idea behind CGP is to create separate plan graphs for each possible world. The expansion phase of Graphplan is changed so that it 1) adds aspects to the plan graph for each possible world separately, and 2) further splits the possible worlds when aspects with uncertain outcomes are added to a graph.

The solution extraction phase of Graphplan tries to achieve the goal in all possible worlds. To preserve soundness, it must consider interactions between aspects chosen in the different possible worlds. More precisely, it must use confrontation whenever an aspect chosen in one world induces an undesirable aspect in another.

Finally, additional mutex relationships can sometimes be inferred between aspects in different possible worlds (which leads to mutex relationships between propositions in different possible worlds). As shown in the results section, these induced mutex relationships significantly improve the performance of the solution extraction process.

Our objective in developing CGP was to see if the impressive performance of Graphplan on STRIPS planning problems would extend to planning problems involving uncertainty. Our experiments indicate that the answer is yes. CGP performs significantly better than previous conformant planners. However, there are some drawbacks and limitations to our approach.

First of all, although the possible worlds mechanism is conceptually clear, it is also cumbersome. As the amount of

uncertainty grows, the number of possible worlds grows exponentially and performance deteriorates. To fix this, we would like to confine the representation of uncertainty to only those propositions that we are uncertain about. In the bomb in the toilet problem, if Dunk clogs the toilet unconditionally, there should never be any uncertainty about whether the toilet is clogged or not. We would therefore like to avoid duplicating clogged and ¬clogged in all the possible worlds. Second, we would like to be able to treat independent sources of uncertainty independently, without generating the cross product of the possible worlds. Although we have experimented with some ways of doing this, we have not yet found a completely satisfactory solution. There appear to be difficult trade-offs here; if we use a more compact representation, we loose the ability to capture some of the mutual exclusion relationships, which degrades performance of the solution extraction phase.

A second limitation of CGP is that it does not take probability information into account. We believe that this is not an inherent limitation. Optimistic probabilities could be calculated and stored with each proposition at a level during the expansion phase. Solution extraction should not be attempted at a level unless all the goal propositions appear at that level with probabilities above the desired thresholds. The actual probabilities for a plan could be computed during the solution extraction phase. The Graphplan framework also seems natural for multiple support (Kushmerick, Hanks and Weld 1995), since all ways of achieving a proposition are represented at a level.

Finally, the possible worlds approach taken here can also be used to do sensory and contingent planning. We have done this, and the technique and results are described in (Weld, Anderson and Smith, 1998).

Acknowledgments

Thanks to Mark Peot for providing an initial Lisp implementation of Graphplan, and for providing and assisting with UDTPOP. Thanks to Keith Golden, Mark Friedman, Pandu Nayak, and anonymous reviewers for comments on earlier versions of this paper. This research was supported by Office of Naval Research Grants N00014-94-1-0060 and N00014-98-1-0147, by National Science Foundation Grant IRI-9303461, by ARPA / Rome Labs grant F30602-95-1-0024, by a gift from Rockwell Palo Alto Research Lab, and by NASA Ames Research Center. This work was initiated while the first author was a visiting scholar at University of Washington.

References

Anderson, C., Smith, D. and Weld, D. 1998. Conditional effects in Graphplan. In *Proc. 4th Int. Conf. AI Planning Systems*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proc. Int. Joint Conf. AI*, 1636–1642.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):281–300.

Draper, D. Hanks, S., and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Int. Conf. AI Planning Systems*, 31–36.

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. Principles of Knowledge Representation and Reasoning*, 115–125.

Gazen, B., and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. 4th European Conf. on Planning*, 223–235.

Golden, K. and Weld, D. 1996. Representing sensing actions: the middle ground revisited. In *Proc. 5th Int. Conf. Principles of Knowledge Representation and Reasoning*, 174–185.

Golden, K., Etzioni, O., and Weld, D. 1994. Omnipotence without omniscience: efficient sensor management for planning. In *Proc. 12th National Conf. on AI*. 1048–1054.

Goldman, R. and Boddy, M. 1994. Conditional linear planning. In *Proc. 2nd Int. Conf. AI Planning Systems*, 80–85.

Goldman, R. and Boddy, M. 1996. Expressive planning and explicit knowledge, In *Proc. 3rd Int. Conf. AI Planning Systems*, 110-117.

Koehler, J., Nebel, B., Hoffmann, J. and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. 4th European Conf. on Planning*, 275–287.

Kushmerick, N., Hanks, S., and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1–2):239–286.

McDermott, D. 1987. A critique of pure reason. *Computational Intelligence* 3:151–160.

Penberthy, J. and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. Principles of Knowledge Representation and Reasoning*, 103–114.

Peot, M. 1998. Decision-Theoretic Planning. Ph.D. Dissertation, Dept. of Engineering-Economic Systems, Stanford University.

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning, In *Proc. 1st Int. Conf. AI Planning Systems*, 189-197.

Pryor, L. and Collins, G. 1996. Planning for contingencies: a decision-based approach. *J. Artificial Intelligence Research* 4:287–339.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E. and Blythe, J. 1995. Integrating planning and learning: the prodigy architecture. *J. Experimental and Theoretical AI* 7:81–120.

Warren, D. 1976. Generating Conditional Plans and Programs. In *Proc. Summer Conf. on AI and Simulation of Behavior.*

Weld, D., Anderson, C. and Smith, D. 1998. Extending Graphplan to handle uncertainty & sensing actions. In *Proc. 15th National Conf. on AI.*