Extending Graphplan to Handle Uncertainty & Sensing Actions*

Daniel S. Weld **Corin R. Anderson**
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, WA 98195–2350 USA
{weld, corin}@cs.washington.edu

David E. Smith
Nasa Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035 USA
de2smith@ptolemy.arc.nasa.gov

Abstract

If an agent does not have complete information about the world-state, it must reason about alternative possible states of the world and consider whether any of its actions can reduce the uncertainty. Agents controlled by a contingent planner seek to generate a robust plan, that accounts for and handles all eventualities, in advance of execution. Thus a contingent plan may include sensing actions which gather information that is later used to select between different plan branches. Unfortunately, previous contingent planners suffered defects such as confused semantics, incompleteness, and inefficiency. In this paper we describe SGP, a descendant of Graphplan that solves contingent planning problems. SGP distinguishes between actions that sense the value of an unknown proposition from those that change its value. SGP does not suffer from the forms of incompleteness displayed by CNLP and Cassandra. Furthermore, SGP is relatively fast.

1 Introduction

Classical planners make the unrealistic assumption that the agent has complete information about the initial state of the world. Reactive systems and agents that interleave planning and execution, on the other hand, have imperfect look-ahead and can get caught by irreversible actions. Contingent planning — the generation of a plan whose course of action varies based on the information gained from executing sensing actions — has the potential to yield robust plans even in the face of uncertainty. The prospect of contingent planning is especially desirable in high stakes domains where even an improbable mistake could prove costly. Unfortunately, previous contingent planners have failed to attract widespread adoption due to serious liabilities:

- They often fail to distinguish between sensory and causal effects of actions, or treat sensing explicitly.
- They are often unable to generate a successful plan if sensory actions are unavailable — even if such a plan exists.
- Previous contingent planners are slow.

This paper presents the SGP contingent planning algorithm, an extension of Graphplan [Blum & Furst, 1997] that handles planning problems with uncertainty in the initial conditions[1] and with actions that combine causal and sensory effects. Although we have not completed a formal proof, we believe SGP is complete; certainly, it finds plans missed by CNLP [Peot & Smith, 1992] and Cassandra [Pryor & Collins, 1996]. While contingent planning will always be harder than corresponding problems graced with complete information, SGP solves small problems quickly — a dramatic improvement over previous contingent planners, whose performance was so abysmal that execution times were always omitted from publications.

2 Background: Conformant Graphplan

Since SGP is based on previous planners, we briefly summarize the Graphplan algorithm [Blum & Furst, 1997] as extended to handle conditional effects [Gazen & Knoblock, 1997] and conformant planning, *i.e.*, uncertainty but no sensing actions [Smith & Weld, 1998].

*We thank Mark Peot who provided an initial Lisp implementation of basic Graphplan. Our paper was improved by discussions with Marc Friedman, Keith Golden, Rao Kambhampati, Alon Levy, and Mike Perkowitz . This research was funded by Office of Naval Research Grant N00014-98-1-0147, by National Science Foundation Grant IRI-9303461, and by ARPA / Rome Labs grant F30602-95-1-0024.

[1]Space precludes description of a direct method of handling actions with uncertain effects (see [Smith & Weld, 1998]), but they can be modeled as actions whose deterministic effects are conditional on hidden state variables defined in the initial state.

Graphplan, in its original form, solves STRIPS planning problems in a deterministic, fully-specified world. Both the preconditions and effects of its action schemata are conjunctions of literals (*i.e.*, denoting 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 then performs a backward-chaining search for an actual solution; if no solution is found, the cycle repeats.

2.1 Graph Expansion

The planning graph contains two types of nodes, proposition nodes and action nodes, arranged into levels. Even-numbered levels contain proposition nodes (*i.e.*, ground literals), 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 one such 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, and directed edges connect action nodes to subsequent propositions made true by the action's effects. Graphplan defines a binary mutual exclusion relation ("mutex") between nodes in the same level as follows:

- 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 $i-1$.
- Two propositions at level i are mutex if all ways of achieving the propositions (*i.e.*, actions at level $i-1$) are mutex.

2.2 Solution Extraction

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 (backtrack point) an action a at level $i-1$ that achieves the goal. 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. Otherwise, if backtracking fails, then Graphplan extends the planning graph and tries again.

2.3 Conditional Effects

Recently, several authors have described methods that allow Graphplan to handle operators with conditional effects. The simplest approach breaks operators with conditional effects into a number of separate operators (by considering all minimal consistent combinations of antecedents in the conditional effects) [Gazen & Knoblock, 1997]. To illustrate, consider the following simple model of the action of taking medication:

Medicate:	pre:	eff: (when I ¬I) (when ¬H D)

in which I means the patient is Infected, H means he is Hydrated, and D means he is Dead. If the patient is Infected before the action then he will no longer be Infected afterwards: ¬I. But if the patient takes medication when he is not Hydrated (*i.e.*, ¬H), then the result is Death. This ADL operator is expanded into the following 4 disjoint[2] STRIPS operators, called *aspects* of the ADL original:

Med_1:	pre: (and I H)	eff: ¬I
Med_2:	pre: (and I ¬H)	eff: (and ¬I D)
Med_3:	pre: (and ¬I H)	eff:
Med_4:	pre: (and ¬I ¬H)	eff: D

Note that Med_3 has no effects; we count it as an aspect to facilitate subsequent bookkeeping.

2.4 Conformant Planning

For simplicity, we only describe the Graphplan extensions required to handle uncertainty in the initial conditions; [Smith & Weld, 1998] explains how to handle actions whose effects are uncertain as well. In classical Graphplan, the initial conditions are specified as a complete conjunction of positive and negative literals. To model uncertain initial conditions, one provides a set of w possible worlds, each modeled as a conjunction of literals. The basic idea behind conformant Graphplan (CGP) is to extend a separate planning graph for each possible world (PW), keeping track of mutual exclusion relations across worlds, and then search backwards for a plan that works in all possible worlds.

To illustrate the algorithm, suppose that there is a Drink action with a single effect that makes the patient Hydrated unconditionally. Drink has one aspect:

Drink_1:	pre:	eff:H

[2] Although the different aspects of an action are mutually exclusive, multiple aspects may appear at levels ≥ 3 because proposition levels (except the initial state) can contain both a proposition and its negation.

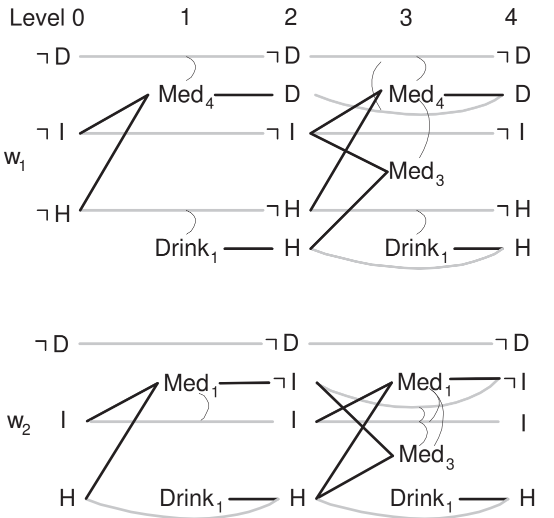

Figure 1: Planning graph for two possible worlds. Gray arcs indicate No-op actions for persistence of propositions from one level to the next. Arcs between two actions denote mutex relationships (to avoid clutter, not all mutexes are shown).

Suppose that initially there are two PWs: $w_1 = \{\neg I, \neg H, \neg D\}$, and $w_2 = \{I, H, \neg D\}$; the goal is to get the patient uninfected and not dead: $\neg I \wedge \neg D$. As we shall see, the best plan in this example is to first Drink (eliminating the interaction between dehydration and subsequent medication in w_1) and then to Medicate. Figure 1 shows the planning graph expanded to level four.

Since both of the goal propositions occur (nonmutex[3]) at level 2 in *each* of the possible worlds, there is the potential of a solution and conformant Graphplan must try to extract a solution.

CGP considers each PW in turn. In w_1 both goals are initially true and hence are carried through by No-op actions. In w_2 $\neg D$ is also made true by a No-op while $\neg I$ can only be achieved by Med_1. Since there are no mutex relations between any of these actions one might think that we have found a solution, but an additional check is necessary. If we choose to perform an action, we must consider its effects in *all* worlds, not just the specific PW in which the action was added. This is because (without sensing) CGP doesn't know which of the worlds it is really in and therefore cannot confine the effects of an action to a single PW. 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 PW. Our example illustrates these complications. The decision to execute Medicate in w_2 (which manifests as Med_1) means that some aspect of Medicate will execute in w_1 as well: Med_4, in fact. Thus, CGP must consider the effect of Med_4 in w_1— which unfortunately includes death, D, in violation of the goal. Since no backtracking alternative remains, CGP concludes that no solution exists at level 2.

Before proceeding, we augment our terminology. We write A:u to refer to aspect A in world w_u. Similarly, we use P:u to refer to proposition P in world w_u. Suppose aspect A:u is present at level i in a PW planning graph and A':v is another aspect of the *same* action at level i in the graph, and v$\neq$u. We say that A:u *possibly induces* A'v, and vice versa. In the example above Med_1:2 possibly induces Med_4:1 at level 1.

Sometimes it is possible to prevent an undesirable, possibly-induced aspect using *confrontation.* To prevent or confront aspect A':v at level i, 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 some other parallel action happened to be executed first, and this parallel action established the precondition of A':v.) Thus if A':v has preconditions $P_1, P_2, \ldots, P_k$ CGP must choose (backtrack point) 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 $\neg P_j$:v

1. Exists at level i, and
2. Is not mutex with any other aspect added during solution extraction at level i.

If so, this No-op is added to the plan, and its precondition $\neg P_j$:v is added to the subgoals at level $i-1$. Note that when doing confrontation, the choice of precondition to confront provides an additional backtrack point.

For an example of confrontation, consider what happens when CGP tries to extract a solution from level 4 of the medication example. At first glance, level 4 appears identical to level 2. In w_1 both goals are present at level 4 solely due to No-op actions, so these are added to the plan. In w_2 there is also no choice: $\neg D$ is made true by a No-op alone, while $\neg I$ can only be achieved by Med_1:2. Again, CGP must consider the effects of the aspects that are possibly induced by Med_1:2, but this time there are two such aspects: harmless Med_3:1 and deadly Med_4:1. As before CGP discovers that Med_4:1 is mutex with the $\neg$D-No-op:1 already in the plan, so Med_4:1 must be confronted, and this decision leads to a choice: which of Med_4:1's preconditions should be negated? Suppose CGP chooses to negate $\neg H$, then CGP adds an H-No-op action to the plan. H-No-op causes H:1 to be a subgoal at level 2 which eventually leads CGP to add $Drink_1$:1 to the plan at level 1. Thus the final, successful plan is: Drink; Medicate — it achieves the goals in both PWs.

[3] Actually it is possible to derive cross-world "induced mutexes" in this example, but while this optimization speeds planning (*e.g.*, it would eliminate the need for searching backwards at level 2), space precludes its presentation; see [Smith & Weld, 1998] for the details.

3 Representing Sensory Actions

For SGP we define a simple but expressive sensor model called SSM. In addition to traditional causal effects, actions have zero or more observational effects, each of which is specified using the syntax (sense *wff*) where *wff* denotes an arbitrary logical expression composed of Graphplan propositions. Thus each sense effect corresponds to a primitive observation that returns one bit of information when executed: T if the logical expression is true in the world immediately prior to execution, and F otherwise.

In order to simplify SGP's handling of sensory actions, we make the following restrictions: sense terms may not be placed in the consequent of conditional effects, and actions with sense effects may not have preconditions. To understand why banning sense terms from the consequents of conditional effects does not limit expressiveness, consider a hypothetical action which does exactly that: (when P (sense Q)). Presumably, the intent behind this statement is for the agent to observe *nothing* if P is false, and to gain information about Q otherwise. Thus, by executing this action, the agent is distinguishing between worlds in which the following mutually exclusive conditions hold: $\neg P, P \wedge Q$, and $P \wedge \neg Q$. In SSM we simply require that this information gain be made explicit with the use of multiple unconditional sense effects. For example, one could achieve the same result with the pair of effects: (sense P) and (sense (and P Q)). The other restriction was that no sensory action could have preconditions; this is of no consequence since every action with preconditions can be rewritten to use conditional effects with the addition of a special *error* proposition.

SSM is capable of representing any noise-free sensor that returns a bounded amount of information. To see this, note that if a sensor returns a bounded amount of information, then its information gain can be encoded in k bits and k sense effects will suffice.

4 Representing an Agent's Knowledge

After executing an action with effect (sense P) the agent knows whether P is true or false, but this form of knowledge is difficult for the planner to exploit. As we shall see, it is much easier to generate a contingent plan if we explicitly represent the agent's ability to discriminate between its PW and the other possibilities. To this end, we define the proposition K¬u:v to mean that if the agent is in w_v then it will know that it is not in w_u.[4]

Consider a domain described in terms of two propositions, P and Q; suppose that we have no knowledge about the initial state, so there are four PWs covering all possible combinations of T and F values. Suppose further that there are two actions: A senses P∧Q and B senses P∨Q. The table below shows the knowledge propositions that result from executing A, B, or both sensory actions.

	Init	After A	After B	After A,B
w_1	P∧Q	K¬2:1 K¬3:1 K¬4:1	K¬4:1	K¬2:1 K¬3:1 K¬4:1
w_2	P∧¬Q	K¬1:2	K¬4:2	K¬1:2 K¬4:2
w_3	¬P∧Q	K¬1:3	K¬4:3	K¬1:3 K¬4:3
w_4	¬P∧¬Q	K¬1:4	K¬1:4 K¬2:4 K¬3:4	K¬1:4 K¬2:4 K¬3:4

Note that each sensor defines a different partition on the possible worlds. If the agent is in w_1 and executes A then it will know that it is not in *any* of the other worlds, but if the agent is in w_2, A will only tell it that it is *not* in w_1.

5 SGP Graph Expansion

Like its ancestor Graphplan, SGP is composed of graph expansion and solution extraction phases. SGP's expansion phase is based on that of CGP, with one major difference: SGP calculates the set of possible knowledge propositions for level $i+1$ from each action's logical sensor definitions and the state of the planning graph at proposition level $i-1$. This calculation is especially tricky when $i \geq 3$ because both proposition P and its negation ¬P might coexist in any given world at proposition levels past the initial state.

For example, suppose that there are two worlds: w_1={¬I, ¬B} and w_2={I, ¬B} where I means that the patient is Infected, and B means that the culture is Blue. There are two actions. Inspect senses B, while Stain has a conditional effect that makes the culture blue exactly when the patient is infected. We show the aspects of Stain below, and Figure 2 shows the planning graph after SGP has extended it to level 4.

Stain_1:	pre: I	eff: B
Stain_2:	pre: ¬I	eff:

Note that executing the sensing action Inspect has no effects at level 1. Since ¬B is true in both w_1 and w_2, the agent receives no useful information when told that B is false. The situation is different, however, when Stain is executed at level 1 and Inspect is executed at level 3. Stain's conditional effect changes the world in a way that allows Inspect to discriminate. As a result level

[4]For readers familiar with possible world semantics, the following translation may help. If the planning domain contains k propositions, then in theory there are 2^k possible worlds. But the planning problem's initial conditions rules out all but w of these. The Kripke structure for the initial state of our planning problem has every remaining PW accessible from all $w-1$ other worlds, because the planner has no additional knowledge of which of the w worlds the agent is in. K¬u:v means that the agent has received information such that w_u is no longer accessible from w_v in the new Kripke structure [Fagin *et al.*, 1995].

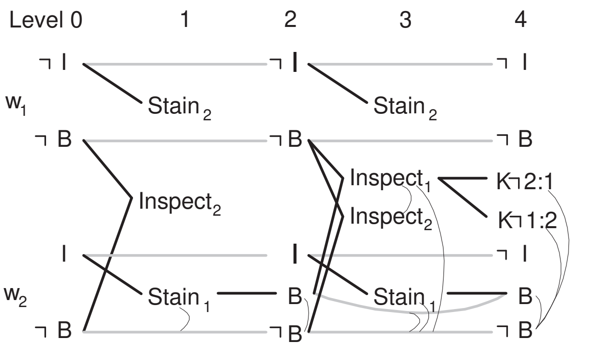

Figure 2: Planning graph showing interaction between causal (Stain) and sensing (Inspect) actions.

4 contains K¬2:1 and K¬1:2; note that these special propositions are supported by a single action instance which links to preconditions in two separate worlds. This type of structure is quite unique to SGP.

Intuitively, the SGP planning graph expansion algorithm considers the set of possible sensor values obtainable in each PW, then takes the cross product to determine the set of partitions which could be induced by the sensor (typically there will be more than one possible partition if causal actions can affect a sensed proposition). These partitions lead to the introduction of K¬u:v propositions at subsequent levels.

Before making this procedure precise, we introduce more terminology. We say a proposition P is *certain in possible world* w_u *at level* i if either P or ¬P is *missing* from the planning graph for w_u at level i. Note that all propositions are certain in all PWs at level 0. In general, a proposition will be certain in a world unless a causal action can change its value. For example, B is certain in w_1 at level 2, but not in w_2 at level 2.

Figure 3 presents the SGP planning graph expansion procedure. Suppose we call it to expand the graph of Figure 2 from level 2 to 4. After SGP adds the purely causal aspects Stain_2:1 and Stain_1:2 to level 3, SGP considers Inspect and iterates over PWs to compute $C = \{\langle\langle \mathsf{F}\rangle, \langle \mathsf{T}\rangle\rangle, \langle\langle \mathsf{F}\rangle, \langle \mathsf{F}\rangle\rangle\}$. Mapping the sensor function (which in this case is the identity function) on the first element of C yields $D = \langle \mathsf{F}, \mathsf{T}\rangle$ shows that the sensor can distinguish the two worlds. Thus SGP adds K¬2:1 and[5] K¬1:2. Note that mapping the sensor function on the other element of C yields $D = \langle \mathsf{F}, \mathsf{F}\rangle$ which shows that the sensor's ability to distinguish is not assured. In particular, the agent must stain the culture before inspecting. Note how the precondition arcs for $\mathsf{Inspect}_1$ in Figure 3 store this information for exploitation during solution extraction.

[5]If sensors are noiseless, then these knowledge propositions will always be symmetric. To see how asymmetric knowledge could be deduced from a noisy sensor, suppose the patient is uninfected in w_1 and infected in w_2, and consider a blood test that might give false positives but never false negatives. One value of D will return F for w_1 and T for w_2, and since the agent knows that the F can't be a false negative it can conclude K¬2:1. The view from w_2 is different, however; the agent can't rule out the possibility of a false positive and hence gains no information.

```
ExpandGraph from prop level i−1 to i+1
  Foreach possible world w_u
    Foreach aspect A of a nonsensory action
      If A's preconditions occur non-mutex at level i−1,
      Then instantiate A at level i, its effects at i+1, and
        link the instance to its preconditions and effects.
  Foreach sensory action, S, with effect (sense φ)
    Suppose φ mentions k atomic propositions {P_1, ..., P_k}.
    Foreach possible world w_u
      Let V_j denote the set of values of P_j:u at level i−1
        ;; I.e. V_j is either {T}, {F}, or {T,F}
      Let C_u be a subset of V_1 × ... × V_k
        by filtering out all mutex combos
        ;; If each prop is certain in w_u at i−1 then |C_u| = 1
        ;; But in the worst case |C_u| = 2^k for each PW w_u
    Let C be a subset of C_i × ... × C_w, by
      filtering out cross-world mutexes
      ;; If each P_j is certain in all worlds, then |C| = 1
      ;; Worst case: |C| = (2^k)^w
    Foreach element, c ∈ C
      ;; c is a w-tuple of k-tuples of boolean values for P_j
      ;; Each k-tuple, thus leads to a value for φ
      Let SA be a new instance of S at level i
      Link the elements of c at i−1 as preconds of SA
      Add any causal effects of SA to level i+1
      Map φ on c getting a boolean w-vector, D
        ;; D = the possibly sensed vals in all w worlds
      For u = 1 to the number of possible worlds w
        For v = u+1 to w
          If D(u) ≠ D(v)
          Then add K¬u:v and K¬v:u to level i+1, and
            link them to SA
  Add all mutex relations for action level i
  Add all mutex relations for prop level i+1
    ;; There are no changes to the mutex rules!
```

Figure 3: Pseudocode for SGP Graph Expansion

6 SGP Solution Extraction

Classical plans, *e.g.* those produced by UCPOP, Graphplan, and CGP, correspond to straight-line programs; each action is executed in sequence. Contingent plans, on the other hand, correspond to programs with if-then-else statements; they can be modeled as an execution tree whose branches refer to information gained by previous sensory actions [Etzioni *et al.*, 1992]. An alternative model, used in C-Buridan [Draper, Hanks, & Weld, 1994] corresponds to a DAG and hence allows plans that rejoin after branches. In this representation each action in the plan is associated with a *context* that specifies the conditions (based on previous observations) under which to execute the action. When an action is first added to the plan, its context is T; conditions get conjoined in response to threats via *conditioning*, the decision not to execute an action in certain worlds [Peot & Smith, 1992].

SGP's solution extraction procedure is a logical generalization of CGP's algorithm [Smith & Weld, 1998]. When SGP adds aspect A:u into the plan in service of goal P:u, it considers all possibly-induced aspects A':v to see if they are mutex with previously added aspects. If A':v does conflict, then there are three choices:

1. SGP can confront A':v, which amounts to forcing a different aspect, A":v, of the action to execute instead, or
2. SGP can condition A':v on a sensor, by finding some sensor that discriminates between w_u and w_v so the agent can execute A in w_u but not in w_v, or
3. SGP can backtrack, *i.e.* retract its decision to support P:u with A:u (or retract an even earlier decision).

6.1 Representing Contingent Plans

From an implementation point of view, however, SGP's solution extraction procedure is very different from CGP's, because SGP represents each level of the partially-specified plan as a dynamic constraint satisfaction problem (CSP) [Mittal & Falkenhainer, 1990], *i.e.* a dynamically-varying set of variables and values with associated constraints. When an aspect of action A is added to level i of a plan with w possible worlds, SGP creates w new CSP variables: $A_1, \ldots, A_w$. The set of possible values of A_u equals the set of aspects of A that are present in the planning graph in w_u at level i *plus* the special value No-exec. Suppose SGP assigns A_1 and A_2 both to equal the value No-exec, and for all $u > 2$ A_u denotes an aspect of A. This assignment of values represents the decision to execute A in all PWs except w_1 and w_2.

Of course, the only way that an agent *could* execute A in all PWs except w_1 and w_2 is if the agent is capable of *distinguishing* between w_1 or w_2 and all other worlds. Hence, the assignment above will lead to knowledge subgoals at level $i-1$. Specifically, the agent must know both K¬u:v and K¬v:u for $v \in \{1, 2\}$ and forall $u > 2$. Note that the agent *need not know which world it is in* (*e.g.*, it need not distinguish between w_1 and w_2); it simply must be able to *rule out* those worlds when in some other world w_u.

6.2 Example

Consider a domain with the Medicate, Stain and Inspect actions defined earlier (but not Drink). Suppose there are two initial worlds: w_1={¬I, ¬H, ¬B, ¬D} and w_2={I, H, ¬B, ¬D}. In other words, the culture is definitely not Blue, the patient is not Dead, it is unknown if the patient is Infected and if he is Hydrated, but these last two propositions are correlated. The goal is to have the patient uninfected and alive: ¬I ∧ ¬D. As we shall see, the best plan is to stain the culture, inspect it, and medicate only if the culture is blue.

```
ExtractSoln(goals G = {..., P:x, ...} at level i)
 If i = 0 and all goals in G are in the appropriate initial PW
 Then return success
 Elseif i = 0 or G contains P:x and Q:y which are mutex
 Then backtrack.
 Let V = {} ; V will be the CSP variables to be solved
 Foreach P:x ∈ G do
   Choose aspect A_j:x (from level i − 1) that produces P:x
     Backtrack if there is no such A_j:x, or
     If ∃v ∈ V such that A_j:x is mutex with all of v's values.
   Let A be the action from which aspect A_j:x was derived
   For each PW w_y create the new CSP variable A_y
     If A_y is not already in V, then add it
       ;; The domain of values for A_y is the set of aspects
       ;; of A in w_y, plus the special value No-exec
   Restrict the domain of A_x to A_j
 Foreach pair of variables A_y, B_z ∈ V
   Let the consistency constraints between A_y and B_z
   be the set of non-mutex pairs of aspects A_j:y, B_k:z
   where A_y and B_z derive from actions A and B
 Foreach action A
   Forall distinct pairs of worlds w_y, w_z
     If K¬y:z is in the planning graph at level i
     Then A_y = No-exec is consistent with all values for A_z
     Else A_y = No-exec is only consistent with A_z = No-exec
 Use any CSP method to choose an assignment ∀v ∈ V
   Backtrack if no assignment exists.
 Let G_{i−2} = {} ; This will hold the set of next goals
 Foreach A_x ∈ V
   If A_x ≠ No-exec
   Then let G_{i−2} = G_{i−2} ∪ the preconds of the value of A_x
   Else foreach A_y ∈ V whose value ≠ No-exec
     Let G_{i−2} = G_{i−2} ∪ {K¬x:y, K¬y:x}
 Recursively call ExtractSoln(G_{i−2}, i − 2)
```

Figure 4: Pseudocode for SGP solution extraction

We now explain how the solution extraction algorithm (Figure 4) derives this plan from the planning graph shown in Figure 5, starting at level 6. In w_1, both goals can only be supported by No-ops, so SGP commits to them. In w_2 SGP has a choice between No-op and Med_1; suppose it chooses Med_1. This spurs the creation of two CSP variables: M_1 and M_2. The domain of M_2 is simply Med_1, but the domain of $M_1 = \{\mathsf{Med}_4, \text{No-exec}\}$. Since Med_4 is mutex with ¬D-No-op, constraint propagation demands that Medicate not be executed in w_1. Note that confrontation is ruled out by the absence of alternative aspects in the domain of M_1 — the only consistent choice is No-exec. This leads SGP to subgoal on K¬2:1 and K¬1:2 at level 4 which eventually causes $\mathsf{Inspect}_1$ to be added from level 3 and Stain_1 to be added from level 1.

7 Experimental Results

SGP is implemented in Allegro Common Lisp 4.3 and accepts variablized action schemata as input. Table 6

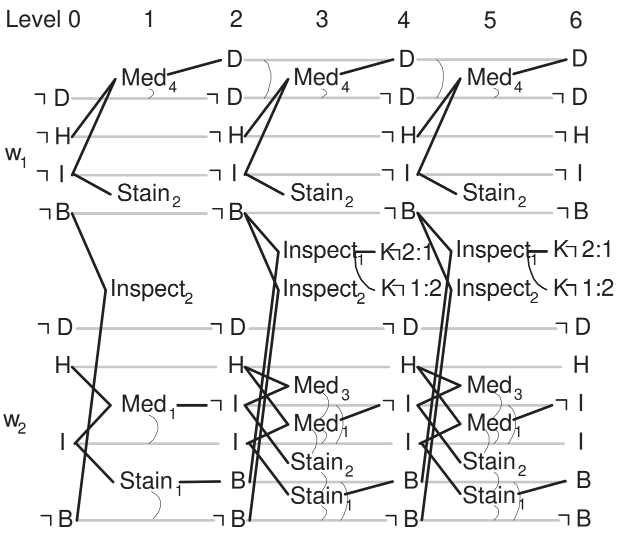

Figure 5: Planning graph for Medical-1 problem.

Name	# PWs	CPU	Plan graph size (a, p)
Medical-1	2	20	(11,6) (18,8) (20,8)
Medical-2	3	40	(30,12) (50,14) (52,14)
Medical-3	4	230	(45,15) (94,18) (100,18)
Medical-4	5	2600	(60,17) (156,21) (168,21)
Ski	3	2152	(57,13) (210,18) (224,19) (227,19) (227,19)
Evanston	2	26	(30, 12) (46, 16) (51, 16) (52, 16)
Fetch-2	4	48	(36, 13) (80, 14) (84, 14)
Coin-1	2	12	(14, 8) (25, 10) (28, 10)

Figure 6: Performance of SGP on sample problems. CPU times are in msec on a 300Mhz Pentium II. For each pair of levels in the planning graph, we show (a, p) — the number of actions and the number of propositions.

shows SGP's performance on a variety of domains, as measured on a 300 MHz Pentium II running Linux. The Medical-1 problem was described in the previous section. In the other medical problems the patient is either healthy or has one of n diseases; there are n separate medicate actions that provide specific cures, but all medications are deadly unless the patient has the corresponding infection. Ski refers to the navigation problem (uncertain road closure) of [Peot & Smith, 1992]. Evanston, Fetch-2, and Coin-1 are from [Pryor & Collins, 1996].

Next, we examined the effect of additional sensory actions on SGP's performance. We chose the "Bomb in the toilet" domain [McDermott, 1987] with five packages and a single toilet. Exactly one of the packages contains a bomb. To defuse the bomb, the offending package must be dunked in an unclogged toilet. Dunking any package will clog the plumbing, so the agent must be careful to dunk correctly. Each sensory action sensed a proposition (*e.g.*, metal-present) that was perfectly correlated with the existence of the bomb in the particular package. As shown in Figure 7, the performance of SGP degrades roughly linearly with the number of sensory actions.

Number of sensory actions	1	2	3	4
CPU time (msec)	400	420	460	500

Figure 7: Performance of SGP on "Bomb in the toilet" domain problems with five packages and one toilet.

# PWs	CPU time (msec)	Plan graph size (a,p)
4	10	(18,8) (34,8)
8	40	(36,12) (84,12)
12	120	(54,16) (150,16)
16	400	(72,20) (232,20)
20	1590	(90,24) (330,24)
24	7450	(108,28) (444,28)

Figure 8: Performance of SGP as a function of uncertainty in the initial state.

In the next experiment (Figure 8), we held the number of sensory actions fixed while we varied the uncertainty of the initial state. We used an artificial domain for this experiment; the goal was to achieve proposition P. As we varied the number of PWs, P was made initially true in half of them. To increase the number of possible worlds, we introduced a number of other unused propositions, exactly one of which was ever allowed to be true.

During our experimentation, we found domains which caused SGP difficulty. The factor limiting SGP seems to be the large number of sensory actions that can be instantiated — worst case 2^{kw} where w is the number of PWs and k denotes the number of atoms in the formula being sensed. In domains where actions sense a single proposition that is certain in most worlds, SGP runs quite quickly. However, when the sensed proposition is not certain in many of the PWs or a complicated formula is being sensed, then SGP creates an exponential number of sensory actions, which slows both graph expansion and solution extraction. Fortunately, most domains can be engineered (*i.e.*, by thinking carefully about the ontology) to avoid these problems.

8 Conclusions

This paper described SGP, an extension of Graphplan that handles uncertain initial conditions and actions with a mixture of causal and sensory effects (as expressed in the SSM sensor language). The central innovations in the SGP algorithm are:

1. An extension to the forward-chaining, graph-expansion phase that derives knowledge propositions from SSM sensor definitions and the previous planning-graph proposition layer.
2. The incorporation of the *conditioning* threat-resolution method into the backward-chaining,

solution-extraction phase.

In addition, SGP generates contingent plans with branches that can rejoin (*i.e.*, a plan is a DAG not a tree). Furthermore, SGP's formulation of solution extraction as a dynamic CSP problem is novel.

Since there might be an exponential number of PWs, contingent planning can be exponentially slower than planning with complete information; however, in domains where mistakes are devastating, the cost may be worth it. Previous papers on contingent planners have neglected to provide quantitative measures of planner speed, instead admitting "less than practical" performance. In contrast, we demonstrate that SGP handles small to medium-sized problems quite quickly.

8.1 Related Work

Space precludes a discussion of work on interleaved planning and execution or the representation of uncertainty; we focus on previous contingent planners. Warplan-C [Warren, 1976] is the patriarch. Universal plans [Schoppers, 1987] and Gapps [Kaelbling, 1988] provide methods for constructing exhaustive contingent plans, but they did not allow explicit description of sensing actions; instead they assumed that complete sensory information would be available at every point during the execution. CNLP [Peot & Smith, 1992] extended SNLP to construct contingent plans. CNLP is notable for its introduction of conditioning, but it assumed free sensing (*i.e.*, it assumed the agent always knew the result of nondeterministic actions once they were performed). UWL [Etzioni *et al.*, 1992] formalized the notion of observational actions using runtime variables, and provided an algorithm for generating contingent plans. [Genesereth & Nourbakhsh, 1993] described techniques for speeding world-space search given incomplete information. Cassandra [Pryor & Collins, 1996] was a fully-implemented contingent planner that distinguished between sensing actions and actions that have uncertain outcomes; in addition Cassandra allowed operators with conditional effects. PLINTH [Goldman & Boddy, 1994] is a total-order planner akin to CNLP. None of these systems could cope with uncertainty if sensing actions were unavailable — even if a working (conformant) plan existed. The probabilistic C-Buridan planner [Draper, Hanks, & Weld, 1994] overcame this limitation and in addition incorporated an elegant sensor model which accounted for noisy sensors. Unfortunately, C-Buridan was so slow that it could not solve the problems listed in this paper.

8.2 Future Work

In the future we hope to augment the SGP sensor model to handle probabilities associated with initial PWs. We would also like to model noisy sensors ala C-Buridan. A different thread would be to extend SGP to handle sensory actions that use runtime variables [Etzioni *et al.*, 1992]. There are many optimizations to be explored, and we seek to run more experiments.

References

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

[Draper, Hanks, & Weld, 1994] Draper, D., Hanks, S., and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Intl. Conf. AI Planning Systems.*

[Etzioni *et al.*, 1992] 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.

[Fagin *et al.*, 1995] Fagin, R., Halpern, J., Moses, Y., and Vardi, M. 1995. *Reasoning about Knowledge.* Cambridge, Mass.: MIT Press.

[Gazen & Knoblock, 1997] Gazen, B., and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. 4th European Conference on Planning.*

[Genesereth & Nourbakhsh, 1993] Genesereth, M., and Nourbakhsh, I. 1993. Time-saving tips for problem solving with incomplete information. In *Proc. 11th Nat. Conf. AI*, 724–730.

[Goldman & Boddy, 1994] Goldman, R. P., and Boddy, M. S. 1994. Conditional Linear Planning. In *Proc. 2nd Intl. Conf. AI Planning Systems*, 80–85.

[Kaelbling, 1988] Kaelbling, L. P. 1988. Goals as parallel program specifications. In *Proc. 7th Nat. Conf. AI.* Morgan Kaufmann.

[Koehler *et al.*, 1997] Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proc. 4th European Conference on Planning.*

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

[Mittal & Falkenhainer, 1990] Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proc. 8th Nat. Conf. AI*, 25–32.

[Peot & Smith, 1992] Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proc. 1st Intl. Conf. AI Planning Systems*, 189–197.

[Pryor & Collins, 1996] Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *J. Artificial Intelligence Research.*

[Schoppers, 1987] Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *Proc. 10th Int. Joint Conf. AI*, 1039–1046.

[Smith & Weld, 1998] Smith, D., and Weld, D. 1998. Conformant Graphplan. In *Proc. 15th Nat. Conf. AI.*

[Warren, 1976] Warren, D. 1976. Generating Conditional Plans and Programs. In *Proceedings of AISB Summer Conference*, 344–354.