Journal of Artificial Intelligence Research () Submitted 07/07/04; published

Planning Graph Heuristics for Belief Space Search

Daniel Bryce DAN.BRYCE@ASU.EDU
Subbarao Kambhampati RAO@ASU.EDU
Department of Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University, Brickyard Suite 501
699 South Mill Avenue, Tempe, AZ 85281

David E. Smith DE2SMITH@EMAIL.ARC.NASA.GOV
NASA Ames Research Center
Comp. Sciences Division, MS 269-2
Moffett Field, CA 94035-1000

Abstract

Belief space search is a technique for solving planning problems characterized by incomplete state information and non-deterministic actions. Planning problems expressed in this space present a much more difficult search problem than classical planning because the space is doubly exponential in the number of boolean state variables. Effective heuristics that reason about the features of belief state space are an, as yet, insufficiently studied topic in planning research. We intend to show how reachability heuristics adapted from classical planning can provide effective search guidance in belief space. Our specific contribution to heuristics for belief space planning is in three respects: (1) defining plan distance between belief states, (2) crafting new heuristics to estimate plan distance, and (3) finding efficient data structures to compute the new heuristics.

We developed two planners to serve as test-beds for our investigation. The first, C*AltAlt*, is a non-observable (conformant) regression planner that uses A* search. The second, $POND$, is a partially-observable (contingent) progression planner that uses LAO* search. C*AltAlt* finds plans as directed paths in the search space whereas $POND$ finds plans as directed cyclic hyper-graphs in the search space. We show the relative effectiveness of our heuristic techniques within these planners. We also compare the performance of these planners with several state of the art approaches in non-deterministic planning.

1. Introduction

Ever since CGP [Smith and Weld, 1998] and SGP [Weld *et al.*, 1998] a series of planners have been developed for tackling conformant and contingent planning problems – including GPT [Bonet and Geffner, 2000], C-Plan [Castellini *et al.*, 2001], PKSPlan [Bacchus, 2002], Frag-Plan [Kurien *et al.*, 2002], MBP [Bertoli *et al.*, 2001a], HSCP [Bertoli *et al.*, 2001b], and KACMBP [Bertoli and Cimatti, 2002]. Several of these planners are extensions of heuristic state search planners that search in the space of "belief states" (where a belief state is a set of states, one of which the agent "believes" it is currently in). Although heuristic search planners are currently among the best, the question of what should constitute heuristic estimates for such planners has not yet been adequately investigated.

Intuitively, it can be argued that the heuristic merit of a belief state depends on at least two factors–the size of the belief state (i.e., the uncertainty in the current state), and the distance of the individual states in the belief state from the goal (belief) state. The question of course is how to compute these measures and which are most effective. We argue that existing planners do not adequately address this question. Our contribution is in a new type of belief space distance measure that more accurately models the mechanics of plans in belief space. The measure accounts for the overlap in deterministic plans that transition the states of a source belief state into the states of a destination belief state.

We will start by discussing, in general terms, what the heuristic estimates *should be* evaluating in belief space planning. This involves formalizing the notions of distances between belief states. Once we figure out what information the heuristics should be computing, we will then turn our attention to ways of computing that information, and finally how to make these computations efficient. We will show that planning graph structures provide a good substrate for the heuristic computation. We start by evaluating non-planning graph approaches – breadth-first search and heuristic search using cardinality. We then try a minimal extension to heuristics used in classical planning by considering heuristics from a single planning graph to guide search. To improve the informedness of the heuristics, we track multiple planning graphs, each corresponding to one of the possible initial states. The number of planning graphs needed is exponential in the number of uncertain initial state literals.[1] Hence, multiple graphs do not scale well as the number of possible initial states grows. The limitations in scaling involve either potentially running out of space to build planning graphs or spending too much time computing heuristics across the multiple planning graphs. Thus, we also describe a significant improvement that addresses these limitations. The idea is to condense the multiple planning graphs to a single planning graph, called a Labelled Uncertainty Graph (LUG). Loosely speaking, this single graph unions the causal support information present in the multiple graphs and pushes the disjunction, describing sets of possible worlds, into "labels".[2] The graph elements are the same as those present in multiple graphs, but much redundancy is avoided. For instance an action that

1. We are able to avoid an exponential number of planning graphs due to non-deterministic effects, ignoring disjunction through treating non-deterministic effects as conjunctive. For example, an effect that gives $a \vee b$, would add both a and b to the subsequent literal layer instead of splitting the planning graph as in CGP [Smith and Weld, 1998].
2. As with multiple planning graphs, in the LUG we ignore the disjunction of non-deterministic effects by treating them as conjunctive. Non-deterministic effects create new possible worlds, but we do not give

was present in all of the multiple planning graphs would be present only once in the LUG and labelled to indicate that it is applicable in a projection from each possible world. We will describe how several powerful planning graph-based heuristics from classical planning, including "level" and "relaxed plan" [Nguyen *et al.*, 2002] can be generalized to the case of multiple planning graphs and the LUG.

An issue in evaluating the effectiveness of heuristic techniques is the many architectural differences between planners that use the heuristics. It is quite hard to pinpoint the global effect of the assumptions underlying their heuristics on performance. For example, GPT is outperformed by MBP–but it is questionable as to whether the credit for this efficiency is attributable to the differences in heuristics, or differences in search engines (MBP uses a BDD-based search). Our interest in this paper is to systematically evaluate a spectrum of approaches for computing heuristics for belief space planning. Thus we have implemented heuristics similar to GPT and MBP and use them to compare against our heuristics developed around the notion of overlap. We implemented the heuristics within two planners, Conformant-$AltAlt$ ($\mathcal{C}AltAlt$) and Partially-Observable Non-Deterministic ($POND$).

Although our main interest in this paper is to evaluate the relative advantages of a spectrum of belief space planning heuristics in a normalized setting, we also compare the best heuristics from this work to existing conformant and contingent planners. Our empirical studies show that planning graph based heuristics provide effective guidance compared to cardinality heuristics as well as the reachability heuristic used by GPT, and our planners are competitive with BDD-based planners such as MBP and GraphPlan-based ones such as CGP and SGP.

The rest of this paper is organized as follows. We present our work by first explaining the state and action representation used within $\mathcal{C}AltAlt$ and $POND$, then discuss appropriate properties of heuristic measures for belief space planning, followed by the set of heuristics used for search control, empirical evaluation, related research, future work, and concluding remarks.

2. Belief State Planners

Our planning formulation uses regression search to find conformant plans and progression search to find conformant and contingent plans. Search is in the space of belief states using actions with conditional and non-deterministic effects. The planning problem is $P = (D, BS_I, BS_G)$ and the domain is $D = (L,\text{S}, A)$, where L is the set of all literals l, S is the set of all states, and A is the set of actions. BS_I and BS_G are the respective initial and goal belief states.

the effects labels to signify the new possible worlds. Essentially, effects can propagate possible world support, but cannot create new possible worlds.

Belief State Representation: As discussed in [Bonet and Geffner, 2000], conformant and contingent planning can be seen as a search in the space of belief states. Given a world state represented in terms of a conjunction of boolean state variables, a belief state BS_i is an arbitrary propositional formula, representing a set of states (also referred to as a set of possible worlds). We consider two special canonical representations of BS_i – clausal representation $\kappa(BS_i)$, which is in CNF (clauses C over literals L), and constituent representation, $\hat{\xi}(BS_i)$, which is in DNF (constituents $\hat{S}$ over literals L). In regression, we're dealing with partial descriptions of belief states, so the constituents of $\hat{\xi}(BS_i)$ may not explicitly represent all states in a belief state. Thus, we define $\xi(BS_i)$ as the *complete* set of states consistent with BS_i. We differentiate states S from constituents $\hat{S}$; states give values to every literal, whereas constituents leave some literals free.

We use the bomb and toilet with clogging problem, BTC [McDermott, 1987], as a running example for this paper. For the uninitiated, here are the arcana of the Bomb in the Toilet family of problems: Bomb in the Toilet (BT) – the problem includes two packages, one of which contains a bomb, and a toilet. The goal is to disarm the bomb and the only allowable actions are dunking a package in the toilet. The variation "bomb in the toilet with clogging" (BTC) says that the toilet will clog unless it is "flushed" after each "dunking" action. The literals encoding the problem denote that the bomb is armed or not (arm), the bomb is in a package or not ($inP1$, $inP2$), and that the toilet is clogged or not ($clog$).

The belief state representation of BTC's initial condition, in clausal representation, is: $\kappa(BS_I) = arm \wedge \neg clog \wedge (inP1 \vee inP2) \wedge (\neg inP1 \vee \neg inP2)$, or in constituent representation: $\hat{\xi}(BS_I) = (arm \wedge \neg clog \wedge inP1 \wedge \neg inP2) \vee (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2)$. BTC's goal state is partial, its clausal representation is: $\kappa(BS_G) = \neg arm$, and its constituent representation is: $\hat{\xi}(BS_G) = \neg arm$. However, its complete set of states: $\xi(BS_G) = \{(\neg arm \wedge clog \wedge inP1 \wedge \neg inP2), (\neg arm \wedge clog \wedge \neg inP1 \wedge inP2), (\neg arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), (\neg arm \wedge \neg clog \wedge \neg inP1 \wedge inP2), (\neg arm \wedge clog \wedge \neg inP1 \wedge \neg inP2), (\neg arm \wedge clog \wedge inP1 \wedge inP2), (\neg arm \wedge \neg clog \wedge \neg inP1 \wedge \neg inP2), (\neg arm \wedge \neg clog \wedge inP1 \wedge inP2)\}$.

Action Representation: A causative action a, of the action set A, is described in terms of (i) an executability precondition ρ_e, and (ii) several conditional effects φ_j of the form ($\rho_j \implies \varepsilon_j$), where the antecedent ρ_j and the consequent ε_j are, in general, formulas. The executability precondition ρ_e, also a formula, of the action must hold for the action to be executable. We define φ_0 as the unconditional effect of an action where by convention $\rho_0 = \top$ and ε_0 is given.

As an example, the actions for BTC are:

$$\begin{aligned} DunkP1 : \{&\rho_e : \neg clog, \\ &\rho_0 : \top \implies \varepsilon_0 : clog, \\ &\rho_1 : inP1 \implies \varepsilon_1 : \neg arm\} \end{aligned}$$

$$\begin{aligned}
&DunkP2 : \{\rho_e : \neg clog,\\
&\qquad\qquad \rho_0 : \top \implies \varepsilon_0 : clog,\\
&\qquad\qquad \rho_1 : inP2 \implies \varepsilon_1 : \neg arm\}\\
&Flush : \{\rho_e : \top,\\
&\qquad\quad \rho_0 : \top \implies \varepsilon_0 : \neg clog\}
\end{aligned}$$

Observation Representation: An observational action a, of the action set A, is described in terms of (i) an executability precondition ρ_e and (ii) a set of observational partition formulas O. Like causative actions, an observational action is only executable when its executability precondition is entailed by the current belief state. Each observational partition formula $o_j \in O$ defines the properties of a distinct outcome of the observation, where each outcome is non-overlapping with others (i.e. $\forall_{i,j}\ o_i \wedge o_j \models \perp$).

As an example, we can add observational actions to BTC to determine if a package contains a bomb:

$$\begin{aligned}
&SniffP1 : \{o_1 : inP1,\\
&\qquad\qquad\quad o_2 : \neg inP1\}\\
&SniffP2 : \{o_1 : inP2,\\
&\qquad\qquad\quad o_2 : \neg inP2\}
\end{aligned}$$

Our observation representation is more expressive than planners like MBP [Bertoli *et al.*, 2001a], which do not allow the observation of formulas, do not have observational preconditions, and require observations to occur after a causative action (similar to the POMDP model). Our observational representation is less expressive than planners like C-Buridan [Draper *et al.*, 1994], which allows observations in the consequents of conditional effects, so that possible sets of observations are conditional.

2.1 Regression

We perform regression in the C*AltAlt* planner to find conformant plans by starting with the goal belief state and regressing it non-deterministically over all relevant actions. An action is relevant for regressing a belief state if (i) its unconditional effect is consistent with the belief state and (ii) at least one effect consequent entails a literal that is present in the constituent representation of the belief state.

Following Pednault [1987], regressing a belief state BS_i over an action a, with conditional effects, involves finding the executability, causation, and preservation formulas. We define regression in terms of clausal representation, but it can be generalized for arbitrary formulas. The regression of a belief state is a conjunction of the regression of clauses in $\kappa(BS_i)$. Formally, the result $BS_{i'}$ of regressing the belief state BS_i over the action a is defined as:[3]

3. Note that $BS_{i'}$ may not be in clausal form after regression (especially when an action has multiple conditional effects).

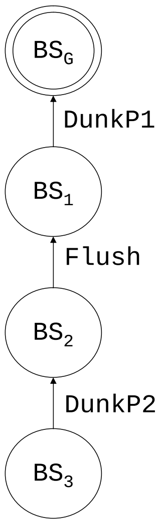

Figure 1: Illustration of regression search for a conformant plan in the BTC problem.

$$BS_{i'} = Regress(BS_i, a) = \Pi_a \wedge \left(\bigwedge_{C \in \kappa(BS_i)} \bigvee_{l \in C} \left(\Sigma_a^l \wedge IP_a^l \right) \right) \tag{1}$$

where

Executability formula (Π_a) is the executability precondition ρ_e of a. This is what must hold in $BS_{i'}$ for a to have been applicable.

Causation formula (Σ_a^l) for a literal l w.r.t all effects φ_j of an action a is defined as the weakest formula that must hold in the state before a such that l holds in BS_i. Formally Σ_a^l is defined as:

$$\Sigma_a^l = l \vee \bigvee_{\substack{j:\varepsilon_j \models l, \\ j \neq 0}} \rho_j \tag{2}$$

Preservation formula (IP_a^l) of a literal l w.r.t. all effects φ_j of action a is defined as the weakest formula that must be true before a such that l is not violated by any effect ε_j. Formally IP_a^l is defined as:

$$IP_a^l = \bigwedge_{\substack{j:\varepsilon_j \models \neg l, \\ j \neq 0}} \neg \rho_j \tag{3}$$

Termination: Regression terminates when search node expansion generates a belief state BS_i which is entailed by the initial belief state BS_I. The plan is the sequence of actions regressed from BS_G to obtain BS_i. For example, in the BTC problem, Figure 1, we have:
$BS_1 = Regress(BS_G, DunkP1) = \neg clog \wedge (\neg arm \vee inP1)$.

The first clause is the executability formula and the second clause is the causation formula for $DunkP1$'s conditional effect and $\neg arm$. Regressing BS_1 with $Flush$ gives:
$BS_2 = Regress(BS_1, Flush) = (\neg arm \vee inP1)$

For BS_2, the executability precondition of $Flush$ is $\top$, the causation formula is $\top \vee \neg clog = \top$ and $(\neg arm \vee inP1)$ comes through persistence in the causation formula. Finally,
$BS_3 = Regress(BS_2, DunkP2) = \neg clog \wedge (\neg arm \vee inP1 \vee inP2)$.

From our example, we terminate at BS_3 because $BS_I \models BS_3$. The plan is $DunkP2$, $Flush$, $DunkP1$.

2.2 Progression

We use progression in the $POND$ planner to generate conformant and contingent plans as (possibly cyclic) hyper-graphs with LAO* search. $POND$ uses the progression function, $Progress(BS_i, a)$, Figure 2, which returns a set of belief states B. When a is not applicable to BS_i or the only element of B is BS_i (meaning a new belief state was not generated) then Progress returns $\emptyset$. Since actions are either strictly causative or strictly observational, the $Progress$ function either generates a single successor or partitions the states of BS_i into several belief states, as outlined in pseudo-code in Figure 2. When a is causative (lines 2-18), we apply a to all states $S \in \xi(BS_i)$ and take the disjunction of all resulting formulas as the result (lines 4-17). The result of applying a to a state S is constructed by taking the conjunction of the consequents ε_j of applicable effects φ_j of a where $S \models \rho_j$ (lines 6-8), then taking the conjunction of it with the persistence literals of S (lines 9-15). When a is observational (lines 19-24), $Progress$ returns a set of successors, where each element is the conjunction of a distinct observational partition $o \in O$ with BS_i.[4] As previously mentioned, we change the result of $Progress$ to $\emptyset$ if applying an action does not result in a new belief state (lines 25-27).

Intuitively, strictly causative actions – which can be non-deterministic – have one successor belief state (requiring a single edge to represent the relation), but observational actions have several successors (requiring a hyper-edge to represent the relation). Since

4. The semantics for observations are that the belief state BS_i is partitioned immediately instead of maintaining the epistemic knowledge, as would a planner like PKSPlan [Bacchus, 2002]. The plan branches after an observation may be identical, simulating a deferred branch after observation, but the contingent plan extraction does not collapse the segments. A more sophisticated extraction of a contingent plan from the policy could take advantage of repeated actions and simulate the deferred branching.

```
$Progress(BS_i, a)$ :
1: $B := \emptyset$
2: **if** $a$ is causative **then**
3:     $BS_{i'} := \perp$
4:     **for** each $S \in \xi(BS_i)$ **do** /* Apply Action to Each State */
5:         $BS_{i''} := \top$
6:         **for** each $\varphi_j$ of $a$, where $S \models \rho_j$ **do** /* Apply Each Applicable Effect */
7:             $BS_{i''} := BS_{i''} \wedge \varepsilon_j$
8:         **endfor**
9:         $P := \top$
10:        **for** each $l \in L$ **do** /* Determine Persistence Literals */
11:            **if** $BS_{i''} \not\models \neg l$ and $S \models l$ **then**
12:                $P := P \wedge l$
13:            **endif**
14:        **endfor**
15:        $BS_{i''} := BS_{i''} \wedge P$
16:        $BS_{i'} := BS_{i'} \vee BS_{i''}$
17:    **endfor**
18:    $B := B \cup BS_{i'}$
19: **else** /* $a$ is observational */
20:    **for** each $o_j \in O$ of $a$ **do** /* Apply Each Observation to Belief State */
21:        $BS_{i'} := o_j \wedge BS_i$
22:        $B := B \cup BS_{i'}$
23:    **endfor**
24: **endif**
25: **if** $\mid B \mid = 1$ and $BS_i \in B$ **then** /* Check if result is productive */
26:    $B := \emptyset$
27: **endif**
28: **return** $B$
```

Figure 2: Progression function pseudo-code.

simple edges are a special case of hyper-edges, during search node expansion we add a set of hyper-edges to the search graph (one hyper-edge for each action). The edges for a single action connect the progressed belief state BS_i to each successor in B; each of these edges must reach a goal in a feasible solution. The hyper-edge for each action applied to a belief state is an “or”-edge (where one hyper-edge of a node is used in a solution), and a hyper-edge is a collection of “and”-edges (where all edges must reach a goal belief state in a solution). This gives rise to an And/Or graph search problem.

2.3 $POND$

Progression planning in $POND$ uses LAO* to find plans. LAO* [Hansen and Zilberstein, 2001] is a search algorithm that generalizes AO* to find solutions with cycles, as required in stochastic shortest path (SSP) problems. In SSP, LAO* has major advantage over standard value or policy iteration algorithms in that it leverages having an initial state to visit only the reachable parts of the state space when finding goal states. Our requirements exactly match the properties of LAO*, namely we have an initial node to direct search and would like to find solutions with branches and loops. However, our use of LAO* is slightly different than the formulation for SSP. In $POND$ we assume non-stochastic actions and partial-observability. With partial-observability we cannot assume every state is fully-observable at every step of the plan, hence belief states (as opposed to states) are nodes in the search graph. Next, observational actions (as opposed to non-deterministic actions, as in LAO*) are associated with the hyper-edges in the search graph because observability is localized to the available observational actions. Lastly, non-deterministic actions generate belief states instead of splitting the resulting states into the search graph.

In $POND$, LAO* decides the nodes to expand with $Progress$ and when the search graph contains a solution. LAO* decides a solution exists by first determining that its best solution graph does not contain any unexpanded nodes, and then performing a final set of dynamic programming backups to make sure that the error bound on the solution is acceptable (i.e. a better solution cannot be found).[5] Should the solution contain unexpanded nodes or if the dynamic programming introduces unexpanded nodes, then search continues by expanding the unexpanded nodes. Termination occurs via the error bound’s convergence to zero because a solution without a goal, that is reachable by all paths, will have a non-zero error bound. The reason for a feasible solution’s non-zero error bound is that the dynamic programming will lower the solution’s cost by changing the best actions for nodes until all nodes in the best solution have a path to reach the goal. Dynamic programming chooses best actions based on the rationale that the goal state is the least costly state to reach and the overall value of the solution is minimized when all paths reach the goal. Note, mini-

5. We use an error bound of zero.

mizing the cost of a solution only guarantees an optimal solution when heuristic values for unexpanded nodes are admissible.

The solution exists as subgraph of the search graph, and we extract the solution by following the hyper-edges marked by LAO*. We extract a plan from the search graph by starting at the initial belief state's node (keeping a list of visited nodes and the associated best actions) and traversing the edges of marked hyper-edges depth first. The action associated with each marked hyper-edge is added to the current branch of the plan until the traversal reaches the node of a goal belief state or an already encountered node. When a goal belief state is encountered, recursion backs up to the last node whose hyper-edge has unexplored edges, and new branch is started. Should the traversal encounter a node already visited, then a goto action is added to the plan, that refers to the action of the re-encountered node. In this manner we are able to extract, in the most general case, cyclic contingent plans. For example, $POND$ generates the following cyclic contingent plan for the omelette problem (detailed in Appendix A):

```
1: (grab)
2: (break_egg large)
3: (inspect large)

     IF: (NOT n0bads_large)
     4: (clean large)
     GOTO 1

     IF: n0bads_large
     DONE
```

As an example of search in $POND$, consider the BTC example (Figure 3) with observational actions. Applying actions to the initial belief state, we get:

$BS_4 = Progress(BS_I, DunkP1)$
$\quad = (inP1 \wedge \neg inP2 \wedge clog \wedge \neg arm) \vee (\neg inP1 \wedge inP2 \wedge clog \wedge arm)$,

and

$B_1 = Progress(BS_I, SniffP1) = \{BS_5, BS_6\}$
$\quad = \{(inP1 \wedge \neg inP2 \wedge \neg clog \wedge arm), (\neg inP1 \wedge inP2 \wedge \neg clog \wedge arm)\}$.

Say we explore BS_4, then it can be progressed further to form:

$BS_7 = Progress(BS_4, Flush)$
$\quad = (inP1 \wedge \neg inP2 \wedge \neg clog \wedge \neg arm) \vee (\neg inP1 \wedge inP2 \wedge \neg clog \wedge arm)$,

and we can finish the plan with:

$BS_8 = Progress(BS_7, DunkP2)$
$\quad = (inP1 \wedge \neg inP2 \wedge \neg clog \wedge \neg arm) \vee (\neg inP1 \wedge inP2 \wedge \neg clog \wedge \neg arm)$.

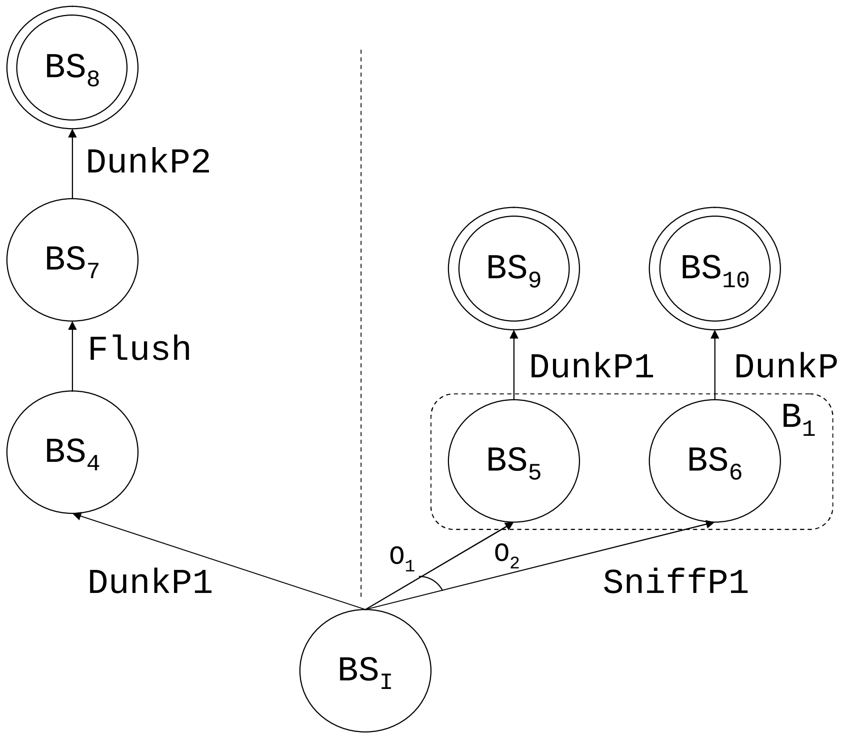

Figure 3: Illustration of progression search for a conformant plan (left) and a contingent plan (right) in the BTC problem.

BS_8 is a terminal belief state because its formula entails the goal belief state.

Alternatively, consider if we expanded BS_5 and BS_6 instead of BS_4. We could find:
$BS_9 = Progress(BS_5, DunkP1) = inP1 \wedge \neg inP2 \wedge clog \wedge \neg arm$,

which is a goal belief state. We could expand further to find:
$BS_{10} = Progress(BS_6, DunkP2) = \neg inP1 \wedge inP2 \wedge clog \wedge \neg arm$,

which is a goal belief state.

A more complex example of the omelette problem, which involves a cycle in the solution, is discussed in Appendix A.

3. Belief State Distance

Before getting into heuristics we must start by discussing what measures are worth estimating. In this section we consider heuristic guidance for belief space planning in the specific case of regression search (pointing out differences in progression search). Consider the example in Figure 4; there are two belief states BS_1 and BS_2 that we are trying to assign heuristic measures for the difficulty of reaching the initial belief state BS_I. We would like

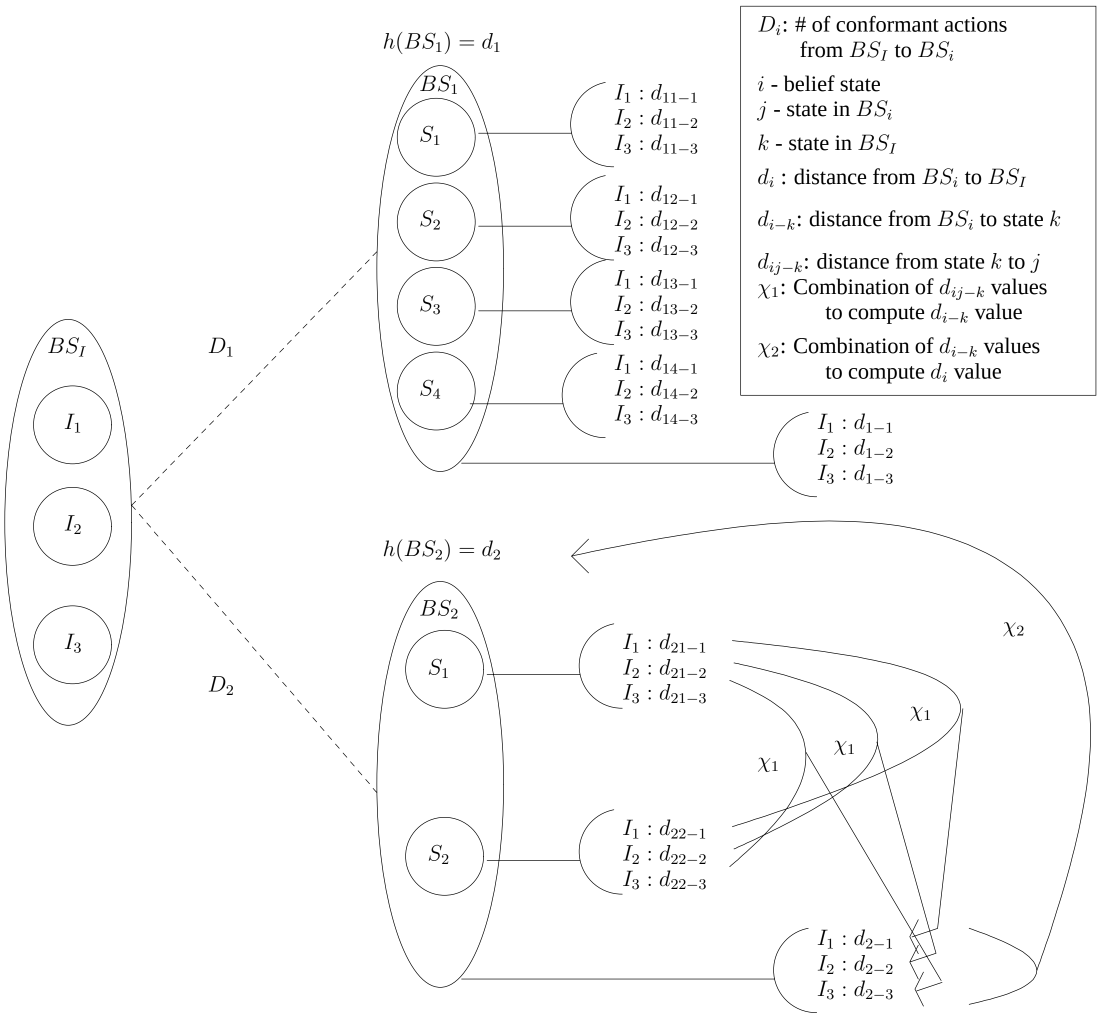
D_i: # of conformant actions from BS_I to BS_i
i - belief state
j - state in BS_i
k - state in BS_I
d_i : distance from BS_i to BS_I
d_{i-k}: distance from BS_i to state k
d_{ij-k}: distance from state k to j
χ_1: Combination of d_{ij-k} values to compute d_{i-k} value
χ_2: Combination of d_{i-k} values to compute d_i value
$h(BS_1) = d_1$
$h(BS_2) = d_2$
BS_I
BS_1
BS_2
D_1
D_2
χ_1
χ_2

Figure 4: Conformant Plan Distance Estimation in Belief Space

to estimate D_1 and D_2, the actual lengths of plans from BS_I to BS_1 and BS_2, respectively. The arcs on BS_2 labelled χ_1 and χ_2 are showing how state distance measures are combined.

There are several factors to consider and leverage in making this estimation of D_1 and D_2:

1. $\xi(BS_i)$, the set of states in the belief state.

2. Reachability measures between pairs of individual states, d_{ij-k}, where each pair is a state S_k from BS_I and S_j from BS_i, as well as χ_1 and χ_2, the combination techniques for the distances of individual states to obtain d_i, a distance estimate to D_i.

3. The *overlap* of independent plans that reach the relevant states of BS_i from states in BS_I.

Preferring belief states with a high cardinality – the number of states in the belief state ($\mid \xi(BS_i) \mid$) – can be a heuristic that assumes for regression that a larger belief state has a higher probability of containing the states of the initial belief state. In progression, a belief state with smaller cardinality may seem preferable because there is less uncertainty about the current state, making planning easier. However, cardinality can be misleading because even though a belief state is large (or small), we may not be able to extend it to include the initial states (or to reach a goal state).

The reachability measures of pairs of states (d_{ij-k}) or pairs of belief states and states (d_{i-k}) also reflect how difficult a plan will be to construct. These d_{ij-k} and d_{i-k} measures can be handled as either numbers estimating the plan length or sets of actions estimating a plan. Also important is how to combine the d_{ij-k} and d_{i-k} measures to ultimately get the estimate for the distance between belief states, d_i. We define two combinations: χ_1, which uses the d_{ij-k}'s to get the d_{i-k} measures, and χ_2, which combines the d_{i-k} measures to get d_i.[6] The operations allowable in χ_1 and χ_2 for numerical estimates are minimum, maximum, and average; and for estimated sets of actions we can take the minimum cardinality set, maximum cardinality set, or the union of sets. Note that the sets of actions can be turned into numerical estimates by taking the cardinalities of the sets; this necessarily happens before we get a final number for d_i.

An important point to note is that in regression not all of the states $S_j \in BS_i$ need to have reachability estimates with respect to each of the initial states, only the min-cost reachable S_j for each $S_k \in BS_I$. Similarly, in progression, with multiple goal states, we only care that each of the states in the current belief state has finite distance to one of the goal states. So we may wish to take the max of the distances from each state to its minimum cost goal state because some of the states in the current state may not be able to reach all of the goal states.

Furthermore, of the reachability measures for $S_j \in BS_i$ there can be much redundancy. The same actions may be used in many of the individual plans that transition the initial states BS_I to BS_i. Hence the plans have high overlap. We define three classes of distance estimates to formalize the idea of overlap:

- **Max S-S:** Maximum state to state distance relates to using the maximum d_{i-k} measure as the estimate. This measure has no notion of overlap.

- **Weak BS-BS:** Weak belief state to belief state distance relates to an aggregation of d_{i-k} measures that considers some overlap (e.g. summing over k for the d_{i-k} measures).

6. Notice that we aggregate over the j states for each k state when deriving the measure for d_i. However, an alternative that aggregates over the k states for each j state is also possible to derive d_i.

- **Strong BS-BS:** Strong belief state to belief state distance relates to a systematic consideration of overlap.

Later, we use these classes to categorize the different planning graph heuristics. As we will show, keeping sets of actions instead of numerical estimates for the d measures will improve our ability to reason about overlap. Planning graphs can aid in finding the sets of actions that improve the measure of plan overlap.

4. Heuristics

Planning graphs serve as the basis for our belief state distance estimation. Planning graphs were initially introduced in GraphPlan [Blum and Furst, 1995] for representing an optimistic, compressed version of the state space progression. The compression lies in unioning the literals from every state at subsequent steps from the initial state. The optimism relates to underestimating how many steps it takes to use actions to support sets of literals. GraphPlan searches the planning graph once the compressed progression (or planning graph) achieves the goal literals in a step's literal list. The search tries to find actions to support the top level goal literals, then find actions to support those actions preconditions and so on until reaching the first step. Planners such as CGP [Smith and Weld, 1998] and SGP [Weld *et al.*, 1998] adapt this idea of compressing the search space with a planning graph by using multiple planning graphs, one for each possible world in the belief state space. CGP and SGP search on these planning graphs, similar to GraphPlan, to find conformant and contingent plans. The work in this paper seeks to apply the idea of extracting search heuristics from planning graphs, previously used in state space search [Nguyen *et al.*, 2002] to belief space search. The basic idea behind using planning graphs for search heuristics is that we can find the first step of a planning graph where a literal in a state appears; this step is a lower bound on the number of actions that are needed to achieve a state with the literal. There are also techniques for estimating the number of actions required to achieve sets of literals. The planning graphs serve as ways to estimate the reachability of state literals and discriminate between the "goodness" of different search states. This work generalizes such literal estimations to belief space search by considering both GraphPlan and CGP style planning graphs.

This section proceeds by describing four sets of heuristics to estimate belief state distance $NG, SG, MG,$ and LUG. NG heuristics are techniques existing in the literature that are not based on planning graphs, SG are techniques based on a single classical planning graph, MG are techniques based on the multiple planning graphs (similar to those used in CGP and SGP) and LUG uses a new labelled planning graph that combines the advantages of SG and MG to reduce the representation size and maintain informedness. Note, we do not include observations in any of the planning graph structures as SGP would, however we do include this feature for future work. The contingent planning formulation directly uses the planning graph heuristics by ignoring observations, and our results show that this still gives good performance.

To illustrate the computation of each heuristic, we use an example from BTC called Courteous BTC ($CBTC$) where a courteous package dunker has to disarm the bomb and leave the toilet unclogged. This problem is used because the goal state has two conjuncts, allowing better illustration of heuristic computation that combines the costs of individual subgoals. The initial belief state of $CBTC$ in clausal representation is $arm \wedge \neg clog \wedge (inP1 \vee inP2) \wedge (\neg inP1 \vee \neg inP2)$, and the goal is $\neg clog \wedge \neg arm$. The optimal action sequences to reach BS_G from BS_I are: $DunkP1, Flush, DunkP2, Flush$, and $DunkP2, Flush, DunkP1, Flush$, thus the optimal heuristic estimate for BS_G, in regression, is $h^*(BS_G) = 4$ because in either plan there are four actions.

The heuristics we evaluate for each of the planning graph structures fall into four categories: max, sum, level, and relaxed plan. These heuristics are well known in classical planning, but require special considerations in the extension to belief space planning, as the following sections illustrate. To help the reader, we outline the major characteristics of these heuristics here. The max and sum heuristics measure the cost of *individually achieving* components of a belief state (literals or clauses) and take the max or sum of these costs. The level heuristic measures the cost of *co-achieving* all components of a belief state. The relaxed plan heuristic measures the number of actions used in a relaxed (backtrack-free) GraphPlan [Blum and Furst, 1995] search to support the belief state.

The heuristics are computed on the three types of planning graph structures (SG, MG, and LUG) and fall into three classes of distance estimates (Max S-S, Weak BS-BS, and Strong BS-BS), as shown in Figure 5. When the graph structure changes, the corresponding heuristics may change the distance measure they are estimating. The rough classes of distance measures that the heuristics fall into are max state to state distance, weak belief state to belief state distance, and strong belief state to belief state distance. Most of the heuristics fall into the max state to state distance class because they are limited by the graph structure or are maximizations by design. Some heuristics fall into the weak belief state to belief state distance measure because they do not take a maximum of state to state distances and do not properly account for all overlap of state to state distances. The strong belief state to belief state distance measure is found by only two heuristics, those that take into account multiple possible worlds and their overlapping plans. As our empirical studies will show, performance is relatively similar for heuristics falling into each of the respective distance measures with variations attributed to the type of graph used for deriving the heuristics.

As in classical planning, in regression search the heuristic estimates the cost of the current belief state w.r.t. the initial belief state and in progression search the heuristic estimates the cost of the goal belief state w.r.t. the current belief state. Thus, in regression

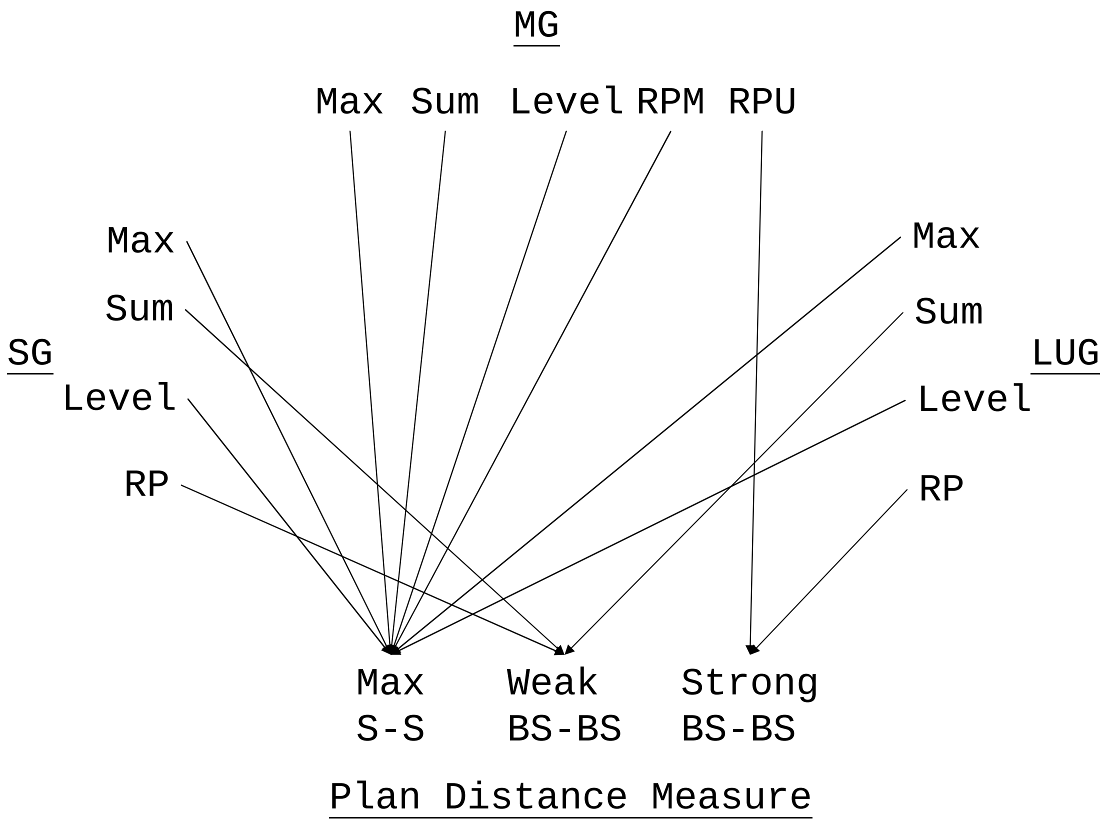

Figure 5: Schematic view of the different heuristics for the various planning graph types and the distance measures they estimate.

search the planning graph(s) are built (projected) once from the possible worlds of the initial belief state, but in progression search they need to be built at each search node (the projected belief state is denoted as BS_P). In the following subsections we describe computing heuristics for regression, but they are generalized for progression by changing the belief state that the planning graph(s) are projected from and the belief state whose cost is estimated.

There are a few techniques which we do not describe or evaluate for progression because of their obvious inefficiencies. Namely, we do not try (i) multiple planning graph heuristics for progression because the number of planning graphs needed in search is $O(2^{2^{2^{|L|}}})$ where L is the set of literals (i.e. the belief space is double exponential in the number of literals, and an exponential number of planning graphs may be needed for each belief state), and (ii) mutex computation techniques because they increase the cost of building planning graphs for every search node.

4.1 Non Planning Graph-based Heuristics (NG)

Breadth First Search The simplest heuristic to search for plans is h_0, where the heuristic value is set to zero.

$$h_0(BS_i) = 0 \tag{4}$$

We mention this heuristic so that we can have a control for evaluating our planners' baseline performance.

Cardinality The idea behind the cardinality heuristic is to count the number of states that are represented by a belief state. This can be useful in regression because the more states in a belief state the better chance that the initial states are in the belief state. In progression fewer states in a belief state means lower uncertainty, hence a better chance of reaching a single goal state. We measure a belief state's cardinality by finding its set of constituents, $\hat{\xi}(BS_i)$ (which approximates $\xi(BS_i)$ in regression[7]), and count them. Formally,

$$h_{card}(BS_i) =\mid \hat{\xi}(BS_i) \mid \tag{5}$$

For instance in $CBTC$, $h_{card}(BS_G) = 1$ because $\neg arm \wedge \neg clog$ is the only constituent of BS_G.

To compensate for regression's partial belief states, we can also measure the number of complete states consistent with the constituent representation :

$$h_{card'}(BS_i) =\mid \xi(BS_i) \mid \tag{6}$$

In $CBTC$, $h_{card'}(BS_G) = 8$ because BS_G has eight states consistent with it. Notice, this may be uninformed for BS_G because some of the states in $\xi(BS_G)$ are not reachable, like $inP1 \wedge inP2 \wedge \neg clog \wedge \neg arm$.

4.2 Single Graph (SG)

The base approach for using planning graphs for belief space planning heuristics is to use a "classical" planning graph by taking all the literals of the projected belief state and inserting each literal individually into the initial layer of the planning graph, ignoring interactions

7. In regression search, the belief states are partial, whereas in progression they are complete, so cardinality varies in its accuracy between these cases.

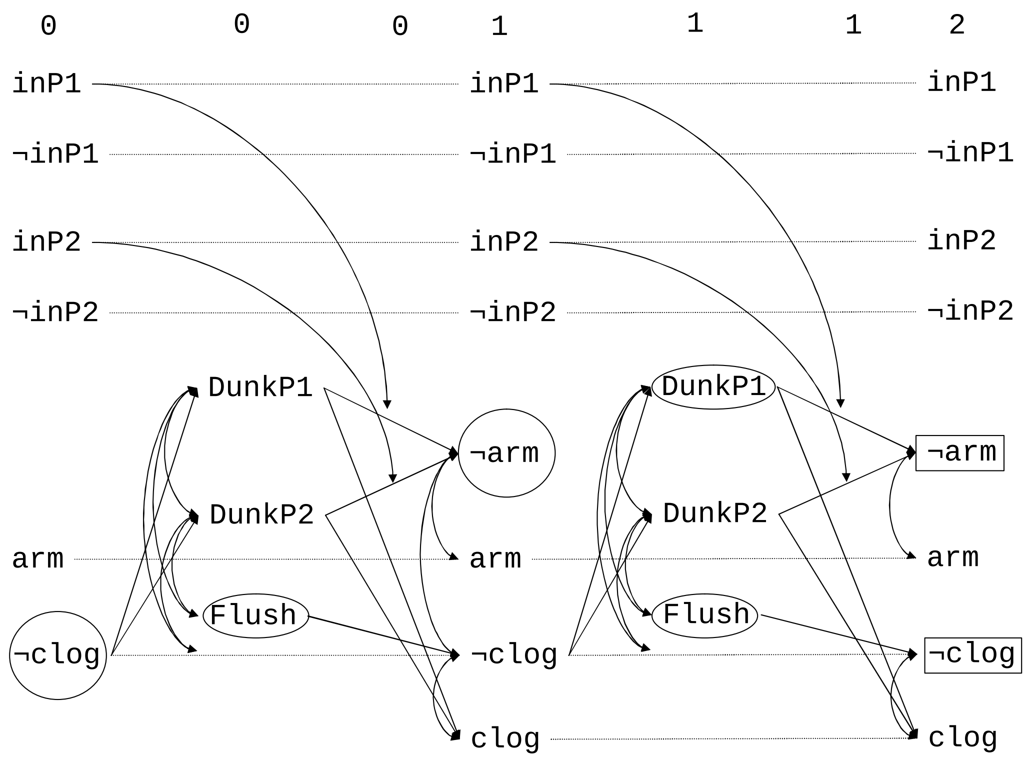

Figure 6: Single planning graph for $CBTC$, with literals used for $h^{SG}_{max}(BS_G)$ and $h^{SG}_{sum}(BS_G)$ circled, literals used for $h^{SG}_{level}(BS_G)$ in boxes, and actions for $h^{SG}_{RP}(BS_G)$ in ovals.

between possible worlds. Graph construction is identical to classical planning graphs (including mutex propagation) and stops when all literals in BS_G are present in a level of the graph.

Thus, for $CBTC$, assuming regression search with BS_I as the projected belief state, the initial level of the planning graph is $\{arm, \neg clog, inP1, inP2, \neg inP1, \neg inP2\}$, ignoring the "xor" connective between $inP1$ and $inP2$ (Figure 6). Once the planning graph is computed, the level lev at which a set of literals appears non pair-wise mutex in the planning graph is later used as the set's $cost$. Notice, the combination χ_2 is not applicable because there is only one d_{i-k} value estimated by a single planning graph.

Max: (h^{SG}_{max}) The first belief space planning heuristic to compute on a planning graph, assuming a clausal representation of belief states, is the max clause achievement cost:

$$h^{SG}_{max}(BS_i) = \max_{C \in \kappa(BS_i)} cost(C) \tag{7}$$

where the cost of a clause is

$$cost(C) = \min_{l \in C} lev(l) \tag{8}$$

Here we estimate χ_1 by finding the cheapest literal as the cost of each clause and taking the max cost clause. This is an underestimate of the most distant state and falls into the max state to state distance measure.

For the $CBTC$ problem we need to find the cost of clauses $\neg arm$ and $\neg clog$, as shown in Figure 6 by the circled literals, to find $h^{SG}_{max}(BS_G)$. The heuristic value is 1 because the max cost clause is $\neg arm$, with cost 1, and $\neg clog$ has cost 0.

Sum: (h^{SG}_{sum}) Another heuristic that can be more aggressive than h^{SG}_{max} is to sum the costs of clauses:

$$h^{SG}_{sum}(BS_i) = \sum_{C \in \kappa(BS_i)} cost(C) \tag{9}$$

It sums the costs of the literals of the closest estimated state in the belief state to estimate χ_1. This measures a weak belief state to belief state distance because the sum accounts for the cost of achieving the clauses of several states independently.

For the $CBTC$ problem we need to find the cost of clauses $\neg arm$ and $\neg clog$, as shown in Figure 6 by the circled literals, to find $h^{SG}_{sum}(BS_G)$. The heuristic value is 1 because the cost of $\neg arm$ is 1, and $\neg clog$ is 0.

Level: (h^{SG}_{level}) A heuristic that maintains the admissibility of the max heuristic but improves the lower bound is to find the level of a set of clauses. The level heuristic is computed by taking the minimum among the $\hat{S} \in \hat{\xi}(BS_i)$, of the first level ($lev(\hat{S})$) in the planning graph where literals of $\hat{S}$ are present with none of them marked mutex. Formally:

$$h^{SG}_{level}(BS_i) = \min_{\hat{S} \in \hat{\xi}(BS_i)} lev(\hat{S}) \tag{10}$$

For the $CBTC$ problem we need to find $lev(\neg arm \wedge \neg clog)$, as shown in Figure 6 by the literals in squares, to find $h^{SG}_{level}(BS_G)$. The heuristic value is 2 because $\neg arm$ and $\neg clog$ do not appear together non-mutex until level 2.

Relaxed Plan: (h^{SG}_{RP}) The level heuristic misses the fact that several actions per step may be needed to support the clauses. To better account for the actual number of actions used, the relaxed plan is computed. We find the relaxed plan to support the maximum level constituent $\hat{S} \in \hat{\xi}(BS_i)$ that contributes to the $h^{SG}_{level}(BS_i)$ and take the number of actions in the relaxed plan as the heuristic value.

The relaxed plan for a belief state BS_i is computed by a backward chaining search on the planning graph. We start at the constituent $\hat{S} \in \hat{\xi}(BS_i)$, such that $lev(\hat{S}) = b =$

$h^{SG}_{level}(BS_i)$. From $\hat{S}$ at level b, for each subgoal $l \in \hat{S}$, a supporting action is selected (ignoring mutexes) from the $b - 1$ action level. We treat persistence of literals and actions equally in supporting literals by arbitrarily selecting either. Once a supporting set of actions ($step_{b-1}$) is determined, the needed preconditions for the actions in $step_{b-1}$ are added to the list of subgoals to support for level $b - 2$. Then, we look at level $b - 2$ for actions. The algorithm repeats until the initial level is reached. Thus, a relaxed plan RP is the set $\{step_0$, ..., $step_i$, ..., $step_{b-1}\}$. Formally, when $h_{level}(BS_i) = b$:

$$h^{SG}_{RP}(BS_i) = \sum_{i=0}^{b-1} \mid step_i \mid \tag{11}$$

For the $CBTC$ problem we find a relaxed plan, as shown in Figure 6 by the actions in ovals, where $h^{SG}_{RP}(BS_G)$ = 3 because the relaxed plan is $\{step_0 = \{Flush\}, step_1 = \{DunkP1, Flush\}\}$. The $step_1$ contains $Flush$ to support $\neg clog$ and $DunkP1$ to support $\neg arm$. Then, in $step_0$ $Flush$ is used to support $DunkP1$'s precondition $\neg clog$. We could have used persistence of literals to get a lower cost relaxed plan, but as stated before, we arbitrarily select persistence or actions. Notice, the relaxed plan does not use $DunkP2$ with $DunkP1$ to support $\neg arm$.

Despite the relaxed plan's inherent status as an estimate, $\neg arm$ is not supported completely because one possible world is not considered. A single, unmodified classical planning graph cannot capture support from all possible worlds. Another disadvantage of single planning graph heuristics are that they make it hard to reason about the *overlap* of independent plans from states in the projected belief state.

4.3 Multiple Graphs (MG)

Single graph heuristics are mostly uninformed because the projected belief state often corresponds to multiple possible states. The lack of accuracy is because single graphs are not able to capture propagation of specific possible world support information. Consider, in $CBTC$ where the projected belief state is BS_I, if $DunkP1$ was the only action we could say that $\neg arm$ is reachable in level 1, but in fact the cost of $\neg arm$ is infinite (since there is no $DunkP2$ to support $\neg arm$ from all possible worlds), and there is no conformant plan.[8]

To account for lack of support in all possible worlds and sharpen the heuristic estimate, a set of planning graphs Γ is considered.[9] Given the projected belief state BS_P, we project a

8. If any of the planning graphs does not "reach" all of the goals, then this is an indication that a conformant plan does not exist.

9. These multiple graphs are similar to CGP's graphs, but lack the more general cross-world mutexes. The mutexes are only computed within each graph, i.e. only same-world mutexes are computed. We also differ from CGP in that we do not create planning graphs for possible worlds created by non-deterministic actions. We treat the disjunctions in non-deterministic effects as conjunctions.

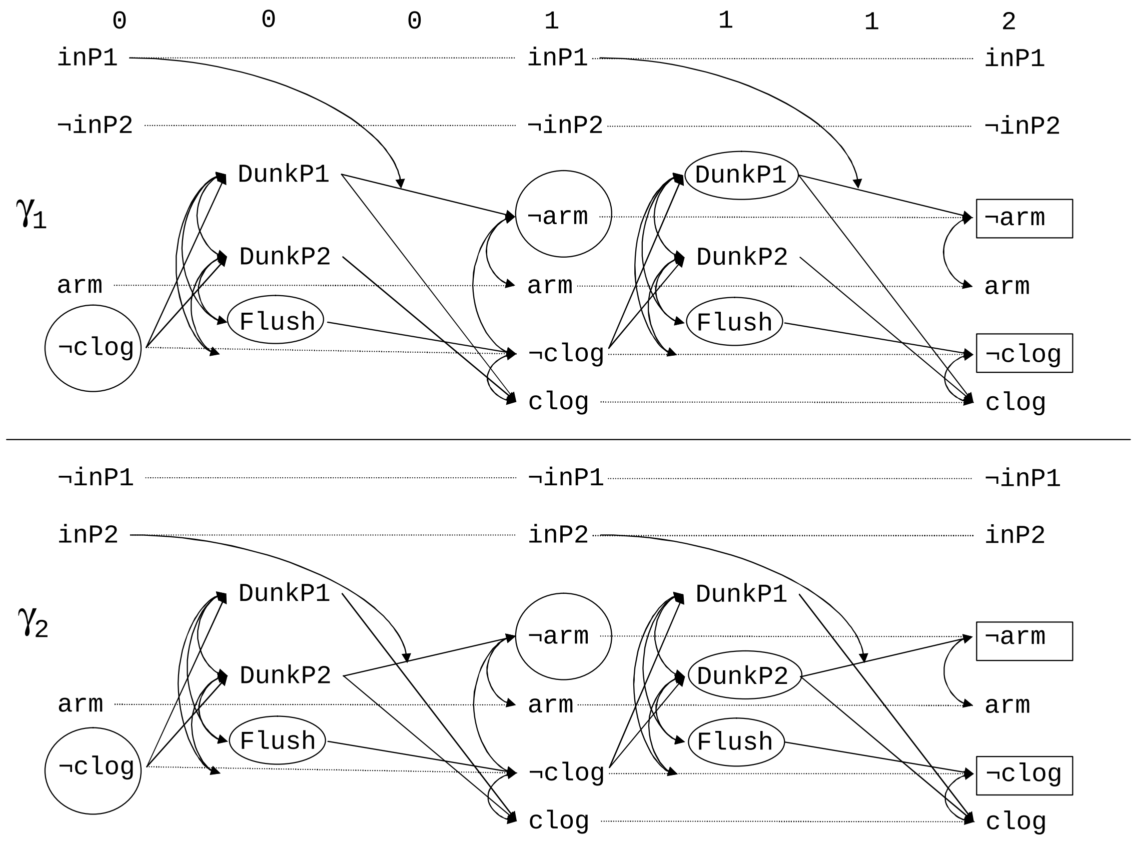

Figure 7: Multiple planning graphs for $CBTC$, with literals used for $h^{MG}_{max}(BS_i)$ and $h^{MG}_{sum}(BS_i)$ circled, literals used for $h^{MG}_{level}(BS_i)$ in boxes, and actions for $h^{MG}_{RP}(BS_i)$ and $h^{MG}_{RPU}(BS_i)$ in ovals.

planning graph $\gamma_k \in \Gamma$ for each constituent of $\hat{\xi}(BS_P)$. With multiple graphs, the heuristic value of a belief state is computed in terms of all the graphs. We now can estimate many d_{i-k} measures and need to define χ_2 methods to combine them.

For example, consider regression search in $CBTC$; there would be two graphs built (Figure 7) for the projected BS_I. They would have the respective initial levels:

$\hat{S}_1 = \{arm, \neg clog, inP1, \neg inP2\}$

$\hat{S}_2 = \{arm, \neg clog, \neg inP2, inP2\}$

In the graph for the first possible world, $\hat{S}_1$, $\neg arm$ comes in only through $DunkP1$ at level 1. In the graph for the second world, $\hat{S}_2$, $\neg arm$ comes in only through $DunkP2$ at level 1. Thus, the multiple graphs show which actions in the different worlds contribute to the same fact's support.

There are several ways to compute the achievability cost of a belief state with multiple graphs, as follows:

Max:(h^{MG}_{max}) The first heuristic to compute with multiple planning graphs is h^{MG}_{max}. The $h^{MG}_{max}(BS_i)$ computes the max cost clause in $\kappa(BS_i)$ for each graph $\gamma_k \in \Gamma$, similar to how $h^{SG}_{max}(BS_i)$ is computed, and takes the maximum. Formally:

$$h^{MG}_{max}(BS_i) = \max_{\gamma_k \in \Gamma} (h^{\gamma_k}_{max}(BS_i)) \tag{12}$$

In this heuristic we estimate χ_1, and χ_2 is a maximum. h^{MG}_{max} considers the minimum cost, relevant literals of a belief state (those that are reachable given a possible world for each graph γ_k) to get d_{i-k} measures. The max is taken because the estimate accounts for the worst (i.e., the plan needed in the most difficult world to achieve the subgoals). This max nullifies the chance of getting any overlap information between the possible worlds. However, using a sum may give a weak estimate of overlap.

For $CBTC$, the goal is $BS_G = \neg clog \wedge \neg arm$. Computing the $h^{MG}_{max}(BS_G)$ for regression (Figure 7) finds $h^{\gamma_1}_{max} = 1$ (denoted by circled facts in the top graph), $h^{\gamma_2}_{max} = 1$ (denoted by the circled facts in the bottom graph), and the max, $h^{MG}_{max}(BS_G) = 1$.

Sum: (h^{MG}_{sum}) The next heuristic with multiple planning graphs is h^{MG}_{sum}. $h^{MG}_{sum}(BS_i)$ computes the sum of the cost of the clauses in $\kappa(BS_i)$ for each graph $\gamma_k \in \Gamma$ and takes the maximum. Formally:

$$h^{MG}_{sum}(BS_i) = \max_{\gamma_k \in \Gamma} (h^{\gamma_k}_{sum}(BS_i)) \tag{13}$$

In this heuristic we estimate χ_1, and χ_2 is a maximum. h^{MG}_{sum} considers the minimum cost, relevant literals of a belief state (those that are reachable given the possible worlds represented for each graph γ_k) to get d_{i-k} measures. As with h^{MG}_{max}, the max is taken because the estimate accounts for the worst. Again, taking a max nullifies the chance of getting any overlap information between the worlds.

From the $CBTC$, the goal is $BS_G = \neg clog \wedge \neg arm$. Computing $h^{MG}_{sum}(BS_G)$ for regression (Figure 7) finds $h^{\gamma_1}_{sum} = 1$ (denoted by circled facts in the top graph), $h^{\gamma_2}_{sum} = 1$ (denoted by the circled facts in the bottom graph), and the max, $h^{MG}_{sum}(BS_G) = 1$.

Level: (h^{MG}_{level}) Similar to h^{MG}_{max} and h^{MG}_{sum}, h^{MG}_{level} is found by first finding $h^{\gamma_k}_{level}$ to get d_{i-k} for each graph $\gamma_k \in \Gamma$, and then taking the max of this value across the graphs. $h^{\gamma_k}_{level}(BS_i)$ is computed by taking the minimum among the $\hat{S} \in \hat{\xi}(BS_i)$, of the first level $lev^{\gamma_k}(\hat{S})$ in the planning graph γ_k where literals of $\hat{S}$ are present with none of them marked mutex. Formally:

$$h^{\gamma_k}_{level}(BS_i) = \min_{\hat{S} \in \hat{\xi}(BS_i)} lev^{\gamma_k}(\hat{S}) \tag{14}$$

and

$$h^{MG}_{level}(BS_i) = \max_{\gamma_k \in \Gamma}(h^{\gamma_k}_{level}(BS_i)) \tag{15}$$

Here we use χ_1 as a minimum to get $h^{\gamma_k}_{level}$, then χ_2 as a maximum for h^{MG}_{level}. Note that this heuristic is admissible. By the same reasoning as in classical planning, the first level where all the subgoals are present and non-mutex is an underestimate of the true cost of a state. This holds for each of the graphs. Taking the max accounts for the most difficult world in which to achieve a constituent of BS_i and is thus a provable underestimate of h^*. GPT's max heuristic [Bonet and Geffner, 2000] is similar to h^{MG}_{level}, but is computed with dynamic programming rather than planning graphs.

For the $CBTC$ goal $BS_G = \neg clog \wedge \neg arm$, computing the $h^{MG}_{level}(BS_G)$ (Figure 7) finds $h^{\gamma_1}_{level} = 2$ (denoted by the level containing facts inside boxes for the top graph), $h^{\gamma_2}_{level} = 2$ (denoted by the level containing facts inside boxes for the bottom graph), and the max, $h^{MG}_{level}(BS_G) = 2$.

Relaxed Plan: (h^{MG}_{RP}) Following the same maximization logic as the other MG heuristics for χ_2, but to account for the actual number of actions used, h^{MG}_{RP} is computed by finding the relaxed plan from the constituent $\hat{S} \in \hat{\xi}(BS_i)$ that contributes to the $h^{\gamma_k}_{level}(BS_i)$ for each $\gamma_k \in \Gamma$ and taking the max of the number of actions across the relaxed plans.

The relaxed plan for a belief state BS_i is computed by a backward chaining search on the planning graph. We start at the constituent $\hat{S} \in \hat{\xi}(BS_i)$, such that $lev^{\gamma_k}(\hat{S}) = b = h^{\gamma_k}_{level}(BS_i)$. From $\hat{S}$ at level b, for each subgoal $l \in \hat{S}$, a supporting action is selected (ignoring mutexes) from the $b-1$ action level. Again, as with the single graph, we arbitrarily select between persistence of literals and actions for supporting literals. Once, a supporting set of actions ($step_{b-1}$) is determined, the needed preconditions for the actions in $step_{b-1}$ are added to the list of subgoals to support for level $b-1$. Then, we support literals at level $b-1$ with actions at $b-2$. The algorithm repeats until the initial level is reached. Thus, a relaxed plan RP_{γ_k} is the set $\{step_{0,\gamma_k}$, ..., $step_{i,\gamma_k}$, ..., $step_{b-1,\gamma_k}\}$. Formally, when $h^{\gamma_k}_{level}(BS_i) = b$:

$$h^{MG}_{RP}(BS_i) = \max_{\gamma_k \in \Gamma}\left(\sum_{i=0}^{b-1} \mid step_{i,\gamma_k} \mid\right) \tag{16}$$

In this heuristic χ_1 is a minimum to get the cheapest estimated relaxed plan for each projected state, then χ_2 is a maximum to get d_i. This gives an inadmissible heuristic for the number of actions to reach the easiest constituent state from the most difficult world.

For $CBTC$, the goal is $BS_G = \neg clog \wedge \neg arm$. Computing the $h^{MG}_{RP}(BS_G)$ (Figure 7) finds $h^{\gamma_1}_{RP} = 3$ ($step_{0,\gamma_1} = \{Flush\}, step_{1,\gamma_1} = \{DunkP1, Flush\}$; actions in ovals for the top graph), $h^{\gamma_2}_{RP} = 3$ ($step_{0,\gamma_2} = \{Flush\}, step_{1,\gamma_2} = \{DunkP2, Flush\}$; actions in ovals for the bottom graph), and the max, $h^{MG}_{RP}(BS_G) = 3$. Notice that this is the closest multiple graph estimate, so far, for $h^*(BS_G)$, but it can be improved by accounting for overlap.

RP-union (h^{MG}_{RPU}): Observing the relaxed plans computed by h^{MG}_{RP} in the $CBTC$ example, we see that the relaxed plans extracted from each graph are different. This information can be leveraged to account for the interaction or overlap of the two possible worlds. Notice, that $step_{1,\gamma_1}$ and $step_{1,\gamma_2}$ contain a $Flush$ action irrespective of which package the bomb is in. Also, $step_{1,\gamma_1}$ contains $DunkP1$, and $step_{1,\gamma_2}$ contains $DunkP2$. Now, taking the union of the two relaxed plans, would give $step_{1,union} = \{DunkP1, DunkP2, Flush\}$, accounting for the actions that are the same between possible worlds and the actions that differ.

In order to get the union relaxed plan, first a relaxed plan is computed for each graph $\gamma_k \in \Gamma$, as in h^{MG}_{RP}. Then, starting from the last step and repeating for each step, we union the sets of actions for each relaxed plan at each level into another relaxed plan. The relaxed plans are *right-aligned*, hence the unioning of steps proceeds from the last step of each relaxed plan to create the last step ($step_{b-1,union}$) of the RP_{union} relaxed plan, then the second to last step for each relaxed plan is unioned for $step_{b-2,union}$ and so on. Then the sum of the numbers of actions of each step in the union relaxed plan (RP_{union}) is used as the heuristic value. Formally, when $h^{MG}_{level}(BS_i) = b$:

$$h^{MG}_{RPU}(BS_i) = \sum_{i=0}^{b-1} \mid step_{i,union} \mid \tag{17}$$

Here χ_1 is a minimum, and χ_2 is a union.

h^{MG}_{RPU} doesn't follow the same form as the rest of the techniques, rather it estimates d_i by finding the relaxed plans corresponding to $\min\limits_j d_{i-j}$ for each graph γ_k, then unions the relaxed plans to get the overlap of plans for relevant states.

The insight of this heuristic is that taking the union of steps of relaxed plans between graphs will account for the same action being used at the same level in multiple worlds. Thus the unioned relaxed plan contains a representative set of *overlapping* actions for achieving the *relevant* states for all source states in the projected belief state.

For the $CBTC$ goal $BS_G = \neg clog \wedge \neg arm$, computing the $h^{MG}_{RPU}(BS_G)$ in regression (Figure 7) finds $RP_{\gamma_1} = \{step_{0,\gamma_1} = \{Flush\}, step_{1,\gamma_1} = \{DunkP1, Flush\}\}$, $RP_{\gamma_2} = \{step_{0,\gamma_2} = \{Flush\}$,$step_{1,\gamma_2} = \{DunkP2, Flush\}\}$, and $RP_{union} = \{step_{0,union} = \{Flush\}$, $step_{1,union} = \{DunkP1, DunkP2, Flush\}\}$. Thus, $h^{MG}_{RPU}(BS_G) = 4$, which is equal to the optimum estimate $h^*(BS_G)$.

4.4 Labelled Uncertainty Graph (LUG)

The multiple graph technique has the advantage of informative heuristics, but the disadvantages of computing redundant support information in different graphs and looking at every graph to compute heuristics. Our next approach condenses the multiple planning graphs to a single planning graph, called a labelled uncertainty graph (LUG), that retains the multiple possible world causal structure. Loosely speaking, this single graph unions the causal support information present in the multiple graphs and pushes the disjunction, describing sets of possible worlds, into "labels" (ℓ). The union of support information is efficiently found by projecting all possible worlds at once, rather than one-by-one. The graph elements are the same as those present in multiple graphs, but represented only once. For instance an action that was present in all of the multiple planning graphs would be present only once in the LUG and labelled to indicate that it is applicable in a projection from each possible world of BS_P.

The worst-case complexity of the LUG is equivalent to the MG representation. The LUG's complexity savings is not realized when the projected possible worlds and the relevant actions for each are completely disjoint; however, this does not often appear in practice – in such cases no conformant plans would exist. The space savings comes in through two aspects: (1) redundant representation of actions and literals is avoided, and (2) labels that facilitate non-redundant representation are stored as BDDs. In practice, the LUG contains the same information as MG with much lower construction and usage costs.

The LUG adds labels to graph elements to symbolically represent which projections through the graph relate to which constituents of BS_P. The labelled elements are each action a, each conditional and unconditional effect relation φ of an action, each literal l, and each mutex relation of the graph. In general, a label is an arbitrary propositional formula describing a set of possible worlds for which a graph element is reachable. The way the LUG is constructed is to label the set of literals in the projected belief state with the possible worlds where they hold and propagate these labels through actions as the graph is built. Construction ends when the goal belief state can be satisfied by literals present in a graph level and the literals are labelled to indicate that the goal belief state is supported by all possible worlds, i.e. the goal is fully-supported. Note, we are constructing the LUG in terms of literals, whereas the search is in terms of formulas. By using literals rather than formulas in the graph layers, we need labels to preserve the disjunction. Later, we describe how to determine if formulas are possibly supported by examining the labels of the formula's literals at levels in the LUG.

The LUG is based on IPP's [Koehler, 1999] planning graph, where there are three layers in a planning graph level: the action layer, effect layer, and literal layer. The extensions are to (i) keep sets of labelled action $\mathcal{A}$, effect relation $\mathcal{E}$, and literal $\mathcal{L}$ layers, and (ii) keep sets of labelled binary mutexes for actions $\hat{\mathcal{A}}$, effect relations $\hat{\mathcal{E}}$, and literals $\hat{\mathcal{L}}$. In

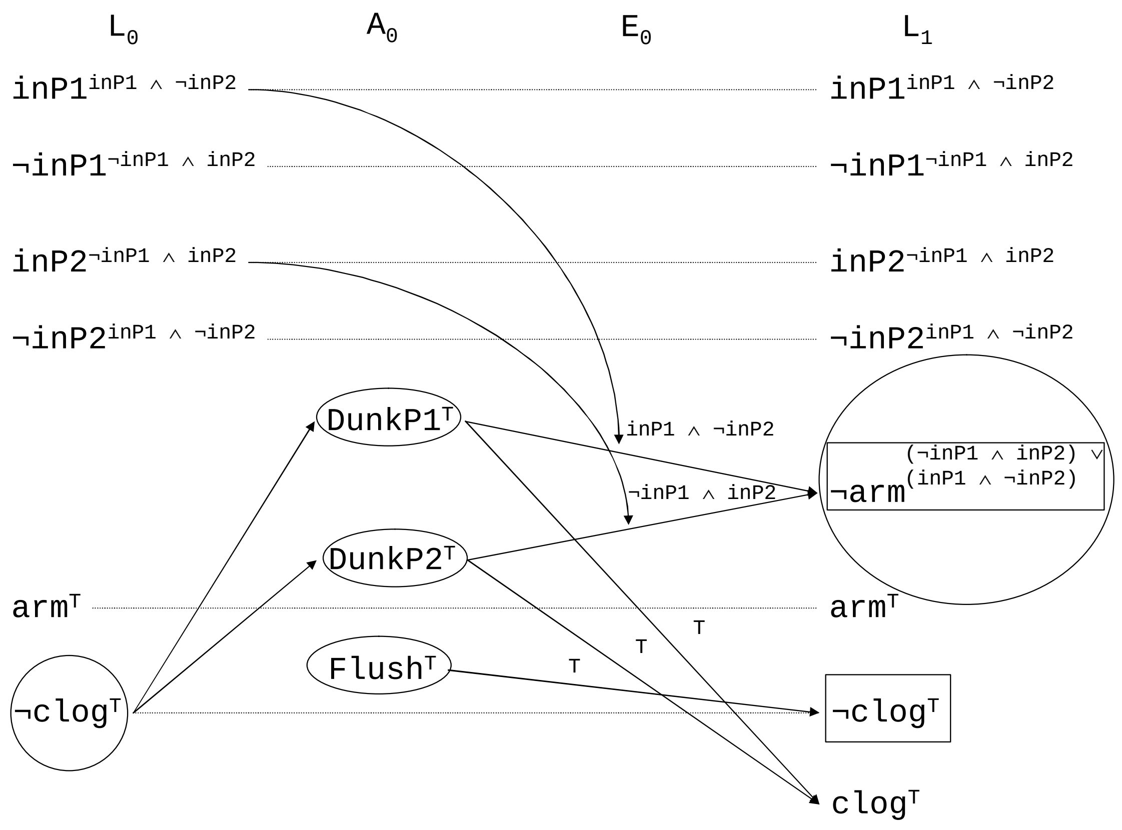

Figure 8: LUG for $CBTC$, with no mutexes. The labels are the superscripts. Literals used for $h^{LUG}_{max}(BS_G)$ and $h^{LUG}_{sum}(BS_G)$ circled, literals used for $h^{LUG}_{level}(BS_G)$ in boxes, and actions for $h^{LUG}_{RP}(BS_G)$ in ovals.

the following subsections, we define the terms fully-supported and supported to aid in the discussion of the label propagation. Then, we describe how to compute mutex relations. Many of the previously mentioned heuristics generalize fairly easily to the LUG, and we end the section by showing how to compute them.

4.4.1 Label Propagation

Recall that a label is a formula describing a set of possible worlds from which a graph element is reachable. Labels (ℓ) are represented as arbitrary propositional formulas and efficient propagation of labels is handled using BDDs. The propagation of labels is based on the intuition that (i) actions and effects are applicable in the possible worlds specified by the conjunction of the precondition literals' labels[10] and (ii) a literal is supported in possible worlds specified by the disjunction of labels of effect relations that support the literal.

We do not propagate labels to account for the non-deterministic outcomes of actions because there is no guarantee that the non-deterministic actions will be used to reach the goals, but it is a requirement that each possible world needs to reach the goals. All effects, including non-deterministic effects, get labels to signify the possible worlds where they are supported, but possible worlds created by non-deterministic effects are not given labels.

During the LUG construction, a common operation is to determine if a formula is fully-supported (i.e. *all* possible world projections can reach a belief state that entails the formula), or supported (i.e. *there exists* a possible world whose projection can reach a belief state that entails the formula). We introduce three notations for support to differentiate how they incorporate mutexes, namely NX for no mutexes, SX for same-world mutexes, and CX for cross-world mutexes. We start with NX to ease the description of label propagation, but describe SX later in this section (and CX in Appendix B). When any of the three support types is applicable, we indicate so with the * symbol.

A formula f is **fully-supported** $(FSp^{NX}(f, k))$ at level k when for all possible worlds S of BS_P, f is supported by S.[11]

A formula f is **supported** $(Sp^{NX}(f, S, k))$ at level k by a possible world S of BS_P when the labels of the literals in f indicate support by S. The labels of literals indicate support by S when the formulas' literals are substituted by their labels at k and S entails the substituted formula. To ease the definition, we consider a canonical form for f, namely a CNF, $\mathcal{C}$, that is supported when:

$$S \models \left(\bigwedge_{C \in \mathcal{C}} \bigvee_{l \in C} \ell_k(l) \right) \tag{18}$$

Here $\ell_k(l)$ is the label of the literal l at level k . Note that full-support is checked for all possible worlds at once by replacing S with BS_P. Note that the LUG is an approximation to the belief space projection from the belief state BS_P, so when we refer to something as supported we mean *possibly* supported.

In Figure 8, $\neg arm$ is not fully-supported at level zero because it is not present (i.e. its label is $\perp$). At level one $\neg arm$ is fully-supported because its label $(inP1 \vee inP2) \wedge (\neg inP1 \vee \neg inP2)$ is entailed by BS_I.

We now describe label propagation, first by showing how to construct the initial literal layer $\mathcal{L}_0$ of the graph, and then showing how a graph level $\{\mathcal{L}_k, \mathcal{A}_k, \mathcal{E}_k\}$ is built.

10. Here we are assuming conjunctive preconditions, but in our formulation we will consider preconditions as general formulas.

11. The notion of fully-supported is a generalization of the level heuristic for classical planning [Nguyen *et al.*, 2002]. It is also similar to the max heuristic used in GPT [Bonet and Geffner, 2000].

$\mathcal{L}_0 \leftarrow$ **insertInitialLiterals**(BS_P)**:**

The initial literal layer $\mathcal{L}_0$ contains all literals l in the problem. The literals not present in BS_P are labelled $\perp$. Each literal l is labelled $\ell_0(l)$ to indicate the set of possible worlds where it holds. The label of literal l is found by the conjunction of l with the formula for the projected belief state BS_P, formally:

$$\ell_0(l) = l \wedge BS_P \tag{19}$$

Assume we are building the LUG once for regression for the remainder of this section. $CBTC$ has the initial layer shown in Figure 8. The known literals (arm and $\neg clog$) are labelled $\top$, and the unknown literals ($inP1$ and $inP2$) are labelled to indicate the possible worlds that contain them.[12] The labels in Figure 8 are the most general formulas to express the possible worlds (to conserve space), but in practice the labels involve all literals deemed necessary by the BDD package. For instance, the label for $inP1$ is denoted in Figure 8 as $inP1 \wedge \neg inP2$, but represents the possible world $inP1 \wedge \neg inP2 \wedge arm \wedge \neg clog$. The label for $inP1$ is found by taking the conjunction of $inP1$ with BS_I, which is $inP1 \wedge arm \wedge \neg clog \wedge (inP1 \vee inP2) \wedge (\neg inP1 \vee \neg inP2)$, reducing to $inP1 \wedge \neg inP2 \wedge arm \wedge \neg clog$. The label for arm is found similarly by taking its conjunction with BS_I, which reduces to BS_I. The reason arm is given the label $\top$, as are all initial level literals that belong to all possible worlds, is due to implementation efficiency and no generality is lost. The fact that $\top$ is entailed by possible worlds that are not part of the projected belief state is not problematic because it suits our purposes that all possible worlds in the projected belief state entail $\top$.

$\mathcal{A}_k \leftarrow$ **insertActions**$(\mathcal{L}_k)$**:**

Once the previous literal layer $\mathcal{L}_k$ is computed, we compute the labelled action layer $\mathcal{A}_k$. $\mathcal{A}_k$ is defined as all applicable actions from the action set A, plus all literal persistence $\diamondsuit l$.[13] An action's executability precondition must be supported for some possible world at level k for the action to be applicable. If applicable, the action's label at level k, using a CNF for the formula of ρ_e, is:

$$\ell_k(a) = \bigwedge_{C \in \rho_e} \bigvee_{l \in C} \ell_k(l) \tag{20}$$

The labels of all the actions in $CBTC$, Figure 8, are $\top$ since the enabling preconditions for all actions are either empty or labelled $\top$.

$\mathcal{E}_k \leftarrow$ **insertEffects**$(\mathcal{L}_k, \mathcal{A}_k)$ **:**

12. As an efficiency measure without loss of generality, we replace the label of every element x with $\top$ if $BS_P \models \ell(x)$.

13. Persistence for a literal l, denoted by $\diamondsuit l$, is represented as an action where $\rho_e = \varepsilon_0 = l$.

The labelled effect relations $\mathcal{E}_k$ depend both on the literal layer $\mathcal{L}_k$ and action layer $\mathcal{A}_k$. $\mathcal{E}_k$ is the set of applicable labelled effect relations at level k. An effect relation is applicable when the associated action is applicable and the antecedent of the effect is supported. The label is the conjunction of the label of the associated action with the label of the formula of the effect's antecedent. The label of an effect i at level k, using a CNF for the formula of ρ_i, is:

$$\ell_k(\varphi_i) = \ell_k(a) \wedge \left(\bigwedge_{C \in \rho_i} \bigvee_{l \in C} \ell_k(l) \right) \tag{21}$$

The conditional effects of the $Dunk$ actions in $CBTC$, Figure 8, have labels to indicate the possible worlds for which they will give $\neg arm$ because their antecedents do not hold in all possible worlds. For example, the conditional effect of $DunkP1$ has the label found by taking the conjunction of the action's label $\top$ with the antecedent's label $inP1 \wedge \neg inP2$, which is $inP1 \wedge \neg inP2$. The other effects have labels $\top$ because they are unconditional and the associated action has label $\top$.

$\mathcal{L}_k \leftarrow$ **insertLiterals**$(\mathcal{E}_{k-1}), k > 0$**:**

The literal layer at k is the set of labelled literals that are added by consequents of effects in $\mathcal{E}_{k-1}$.

A literal is added to the literal layer if it is present in the formula of a consequent of an effect in the previous layer. The label of a literal, $\ell_k(l)$, is the disjunction of the labels of each effect that has the literal in its consequent's formula, using a CNF for the formula of ε_i, it is:

$$\ell_k(l) = \bigvee_{\substack{l \in C, \\ C \in \varepsilon_i, \\ \varphi_i \in \mathcal{E}_{k-1}}} \ell_{k-1}(\varphi_i) \tag{22}$$

The labels of the literals for level 1 of $CBTC$, Figure 8, indicate that $\neg arm$ is fully-supported because its label is entailed by BS_I. The label of $\neg arm$ is found by taking the disjunction of the labels of effects that give it, namely, $inP1 \wedge \neg inP2$ from the conditional effect of $DunkP1$ and $\neg inP1 \wedge inP2$ from the conditional effect of $DunkP2$, to get $(inP1 \wedge \neg inP2) \vee (\neg inP1 \wedge inP2)$. Construction can stop here because BS_I entails the label of the goal $\neg arm$. Equivalently, LUG construction stops when the formula for the goal belief state BS_G is fully-supported.

Same-World Labelled Mutexes There are several types of mutexes that can be found within the LUG. To start with, we only concentrate on those that can evolve from the same possible world because same-world mutexes are more effective as well as easier to understand. In Appendix B we describe how to handle cross-world mutexes, despite their lack of effectiveness in the experiments we conducted.

Same-world mutexes can be represented with a single label, $\hat{\ell}_k(x_1, x_2)$, between two elements (actions, effect relations, or literals). The mutex holds between elements x_1 and x_2 in all worlds S where $S \models \hat{\ell}_k(x_1, x_2)$. We discuss how the labelled mutexes are discovered and propagated for actions, effect relations, and literals.

The use of mutexes requires an extension of the definitions for "supported" and "fully-supported". It is not enough that the labels of literals indicate the appropriate possible worlds support the literals, but also there are no mutexes in each of those possible worlds.

A formula f is **fully-mutex-supported** ($FSp^{SX}(f, k)$) at level k, if it is supported for all possible worlds S.

A formula f is **mutex-supported** ($Sp^{SX}(f, S, k)$) at level k for a possible world S that entails BS_P when (i) the labels of the literals in f indicate f is supported from the possible world S, and (ii) there is no mutex that invalidates f's support. To ease the definition, we consider a canonical form for f, namely a CNF, $\mathcal{C}$. A formula f is inconsistent in a possible world S when there are two clauses where all literals are mutex for S. The formula $\mathcal{C}$ is mutex-supported $Sp^{SX}(f, S, k)$ when:

$$Sp^{NX}(\mathcal{C}, S, k)$$

and

$$\neg\exists_{\substack{C\in\mathcal{C}\\ C'\in\mathcal{C}\\ C\neq C'}} \forall_{\substack{l\in C\\ l'\in C'}} \exists_{\hat{\ell}_k(l,l')\in\hat{\mathcal{L}}_k} S \models \hat{\ell}_k(l, l') \tag{23}$$

The second part of Formula 23 states that there does not exist two different clauses of $\mathcal{C}$ for which all pairs of literals from the clauses are mutex in world S. This assertion means that fully-mutex-supported, unlike fully-supported, cannot be found for all possible worlds by replacing S by BS_P in the above formulas, rather because of mutexes it needs be checked for all S that entail BS_P.

Action Mutexes $\hat{\mathcal{A}}_k$: The action mutexes are a set of labelled pairs of actions. Each pair is labelled with a formula that indicates the set of possible worlds where the actions are mutex. The possible reasons for mutex actions are interference and competing needs.

- **Interference** is a set of labelled mutexes for all pairs of actions interfering because (1) the formula of the unconditional effect consequent of one when taken in conjunction with the formula of either the (a) enabling precondition or (b) unconditional effect consequent of the other is inconsistent, or (2) the conjunction of the formulas from both actions' enabling preconditions is inconsistent. The mutex will exist in all possible world projections, so the label of the mutex is $\top$. Formally, a and a'

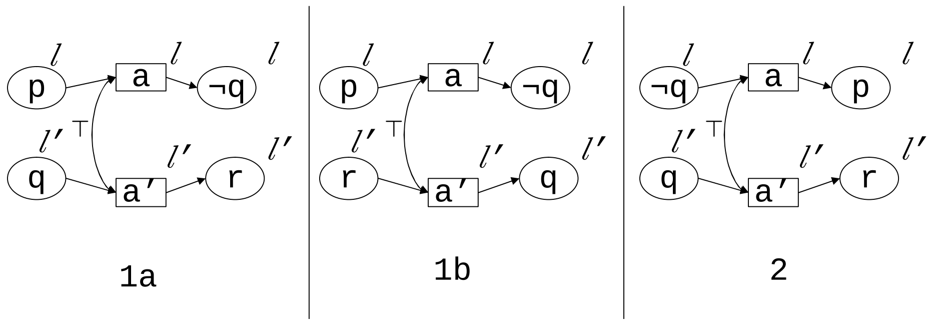

Figure 9: Example of a same-world action interference mutex.

interfere if:

(1a)

$$\varepsilon_0 \wedge \rho'_e \models \perp \; or$$

(1b)

$$\varepsilon_0 \wedge \varepsilon'_0 \models \perp \; or$$

(2)

$$\rho_e \wedge \rho'_e \models \perp \tag{24}$$

Should a and a' interfere, the label of the mutex is:

$$\hat{\ell}_k(a, a') := \top \tag{25}$$

Examples of same-world action interference mutexes are shown in Figure 9. For scenario 1a, actions a and a' are mutex with label $\top$ because a has the unconditional effect $\neg q$ and a' has the enabling precondition q. For scenario 1b, action a is mutex with a' with label $\top$ because a has the unconditional effect $\neg q$ and a' has the unconditional effect q. For scenario 2, a and a' are mutex with label $\top$ because a has enabling precondition $\neg q$ and a' has enabling precondition q.

- **Competing Needs** is a set of labelled mutexes for all actions that have competing needs at level k. This means that there exists a possible world where the conjunction of enabling preconditions is not supported because of a same-world literal mutex ($\neg Sp^{SX}$). The label of the action mutex is the disjunction of all possible world formulas where the conjunction of the pair of actions' enabling precondition formulas are supported, but mutex. The possibly empty label of a mutex between a and a' is:

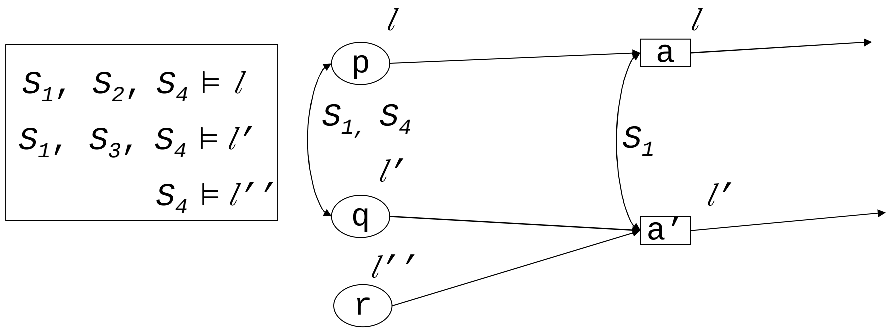

Figure 10: Example of a same-world action competing needs mutex.

$$\hat{\ell}_k(a, a') := \bigvee_{S : \neg Sp^{SX}(\rho_e \wedge \rho'_e, S, k)} S \tag{26}$$

An example of a same-world action competing needs mutex is illustrated in Figure 10. Literal p holds in possible worlds S_1, S_2, and S_4 (denoted by label ℓ), q holds in possible worlds S_1, S_3 and S_4 (denoted by label ℓ'), and r holds in world S_4 (denoted by ℓ''), but p and q cannot hold together in possible worlds S_1 and S_4 because they are mutex in those possible worlds. The action a' has the enabling precondition $q \vee r$. When checking for a competing needs mutex between a and a', we see that the only possible world S where $\neg Sp^{SX}(p \wedge (q \vee r), S, k)$ holds is S_1 because even though q and p are mutex in world S_4, r can support the enabling precondition of a' in S_4 and is not mutex with p.

Effect Mutexes $\hat{\mathcal{E}}_k$: The effect mutexes are a set of labelled pairs of effects. Each pair is labelled with a formula that indicates the set of possible worlds where the effects are mutex. The possible reasons for mutex effects are interference, competing needs, or induced effects. The unconditional effect of an action will become mutex with all unconditional effects of actions where the actions were mutex; futher, these mutexes will be propagated to the conditional effects via the induced mutex rule.

- **Interference** is a set of labelled mutexes for all effects interfering because (1) the formula of the antecedent of one effect when conjoined with the formula of either the (a) antecedent or (b) consequent of the other is inconsistent, or (2) the conjunction of the formulas from both effects' antecedents is inconsistent. The mutex will exist in all possible world projections, so the label of the mutex is $\top$. Formally, φ_i of a and φ'_j of a' interfere if:

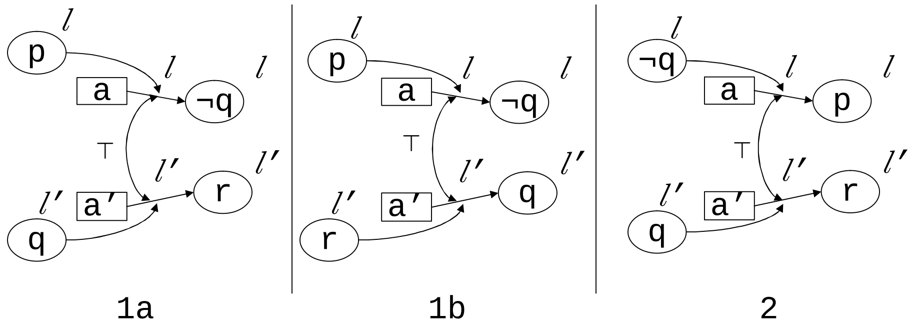

Figure 11: Example of a same-world effect interference mutex.

(1a)

$$\varepsilon_i \wedge \rho'_j \models \perp \; or$$

(1b)

$$\varepsilon_i \wedge \varepsilon'_j \models \perp \; or$$

(2)

$$\rho_i \wedge \rho'_j \models \perp \tag{27}$$

Should φ_i of a and φ'_j of a' interfere, the label of the mutex is:

$$\hat{\ell}_k(\varphi_i, \varphi'_j) := \top \tag{28}$$

An example of a same-world effect interference mutex is shown in Figure 11. For scenario 1a, the conditional effects of actions a and a' are mutex with label $\top$ because the conditional effect of a has the consequent $\neg q$ and the conditional effect of a' has the antecedent q. For scenario 1b, the conditional effect of action a is mutex with the conditional effect of a' with label $\top$ because the conditional effect of a has the consequent $\neg q$ and the conditional effect of a' has the consequent q. For scenario 2, the conditional effects of a and a' are mutex with label $\top$ because a's conditional effect has the antecedent $\neg q$ and the conditional effect of a' has the antecedent q.

- **Competing Needs** is a set of labelled mutexes for all pairs of effect relations having competing needs at level k because the conjunction of their antecedent formulas are not fully-supported at level k. This means that there exists a possible world where the conjunction of effect antecedents is not supported because of a literal mutex. The

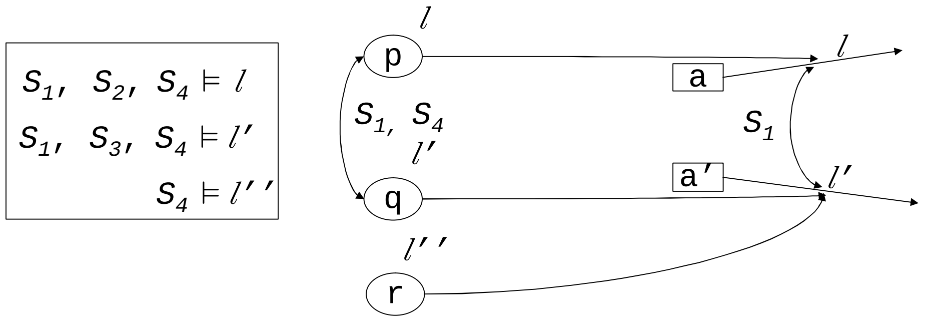

Figure 12: Example of a same-world effect competing needs mutex.

label of the effect mutex is the disjunction of all possible world formulas where the conjunction of the pair of actions' enabling precondition formulas are mutex. The φ_i of action a and effect φ'_j of action a' have the possibly empty label:

$$\hat{\ell}_k(a, a') := \bigvee_{S:\neg Sp^{SX}(\rho_i \wedge \rho'_j, S, k)} S \tag{29}$$

An example of a same-world effect competing needs mutex is illustrated in Figure 12. Literal p holds in possible worlds S_1, S_2, and S_4 (denoted by label ℓ), q holds in possible worlds S_1, S_3 and S_4 (denoted by label ℓ'), and r holds in world S_4 (denoted by ℓ''), but p and q cannot hold together in possible worlds S_1 and S_4 because they are mutex in those possible worlds. The conditional effect of a' has the antecedent $q \vee r$. When checking for a competing needs mutex between a and a', we see that the only possible world S where $\neg Sp^{SX}(p \wedge (q \vee r), S, k)$ holds is S_1 because even though q and p are mutex in world S_4, r can support the antecedent of the conditional effect of a' in S_4 because it is not mutex with p in S_4.

- **Induced** is a set of labelled mutexes that result for effects of each action from having another effect of the same action (i) mutex with effects of other actions, and (ii) the effects of the same action that can be executed together (i.e. induce each other). The simplest example of one effect inducing another is a conditional effect and an unconditional effect inducing each other because any scenario where a conditional effect executes, the unconditional effect must also execute. An effect is induced by another in the possible worlds where they are both supported, or more generally, the effects induce each other in all possible worlds entailed by the conjunction of their labels. Formally, an action a has an effect φ_i that induces effect φ_j, at level k, in the possible worlds described by :

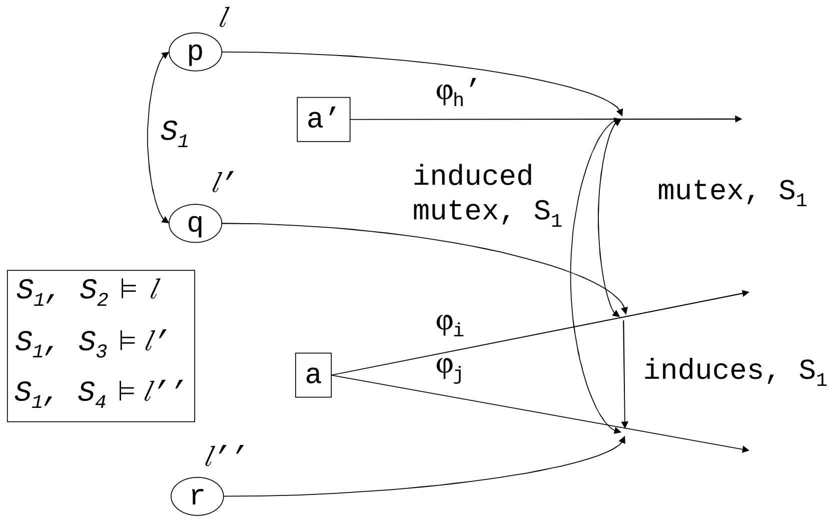

Figure 13: Effect φ_i induces effect φ_j. φ_i is mutex with φ'_h, so φ_j is induced mutex with φ'_h.

$$\ell_k(\varphi_i) \wedge \ell_k(\varphi_j) \tag{30}$$

The mutex is between (a) the effect mutex with the inducing effect and (b) the induced effect. The label of the mutex is the conjunction of the label of the effect that is mutex with the inducing effect, the label of the inducing effect and the label of the induced effect. If φ_i of action a is mutex with φ'_h of action a' in the possible worlds described by $\hat{\ell}_k(\varphi_i, \varphi'_h)$, and φ_i induces effect φ_j of action a, at level k, in the possible worlds described by $\ell_k(\varphi_i) \wedge \ell_k(\varphi_j)$, then there is an induced mutex between φ_j and φ'_h for all possible worlds specified by the conjunction of $\ell_k(\varphi_i) \wedge \ell_k(\varphi_j)$ and $\hat{\ell}_k(\varphi_i, \varphi'_h)$ (see Figure 13). The label of the induced mutex between φ_j and φ'_h, at level k, is:

$$\hat{\ell}_k(\varphi_j, \varphi'_h) := \ell_k(\varphi_i) \wedge \ell_k(\varphi_j) \wedge \hat{\ell}_k(\varphi_i, \varphi'_h) \tag{31}$$

For a more thorough discussion of the methodology behind induced mutexes see [Smith and Weld, 1998].

An example of a same-world induced effect mutex is shown in Figure 13. Literal p holds in possible worlds S_1 and S_2 (denoted by label ℓ), q holds in possible worlds

S_1 and S_3 (denoted by label ℓ'), and r holds in worlds S_1 and S_4 (denoted by ℓ''). Literals p and q are mutex in possible world S_1. The effect φ_i of a induces φ_j in possible world S_1, φ_i is mutex with effect φ'_h of action a' in possible world S_1 because of the mutex between p and q, and φ_j becomes induced mutex with φ'_h in possible world S_1.

Literal Mutexes $\hat{\mathcal{L}}_k$: The literal mutexes are a set of labelled pairs of literals. Each pair is labelled with a formula that indicates the set of possible worlds where the literals are mutex. The reason for mutex literals is inconsistent support.

- **Inconsistent Support** is a set of labelled mutexes for all pairs of literals having inconsistent support in a possible world at level k because each literal is supported, but the conjunction of the literals is not supported in that world at level k because of an effect mutex. The label of the literal mutex at level k is a disjunction of all worlds where the literals are not supported together. The label for a possibly empty inconsistent support mutex between l and l' is:

$$\hat{\ell}_k(l, l') := \bigvee_{S : \neg Sp^{SX}(l \wedge l', S, k)} S \tag{32}$$

An example of a same-world inconsistent literal mutex is shown in Figure 14. Action a has an effect that is supported in worlds S_1 and S_2 (denoted by ℓ) and gives p, action a' has an effect that is supported in worlds S_1 and S_3 (denoted by ℓ') and gives q, and action a'' has an effect supported in world S_4 (denoted by ℓ'') and gives $p \wedge q$. There is an same-world effect mutex between the effects of a and a' in worlds S_1 and S_4. The only possible world S where literals p and q are $\neg Sp^{SX}(p \wedge q, S, k)$ is S_1 because a'' can give support to $p \wedge q$ in S_4.

4.4.2 *LUG* HEURISTICS

The heuristics computed on the LUG capture measures similar to the equivalent heuristics for MG. All of the LUG heuristics estimate d_i directly, thus their combinations χ_1 and χ_2 are also estimates.

Max: (h^{LUG}_{max}) The max heuristic for the LUG finds the maximum across clauses of the literal level $\mathcal{L}_k$ where a clause in $\kappa(BS_i)$ first becomes fully-supported. Formally:

$$h^{LUG}_{max}(BS_i) = \max_{C \in \kappa(BS_i)} \left(\min_{i : FSp^*(C, i)} i \right) \tag{33}$$

For the $CBTC$ problem we need to get the cost of clauses $\neg arm$ and $\neg clog$, as shown in Figure 8 by the circled literals, for $h^{LUG}_{max}(BS_G)$. The heuristic value is 1 because the max level where a clause is fully-supported is 1 for $\neg arm$ and $\neg clog$ is fully-supported at 0.

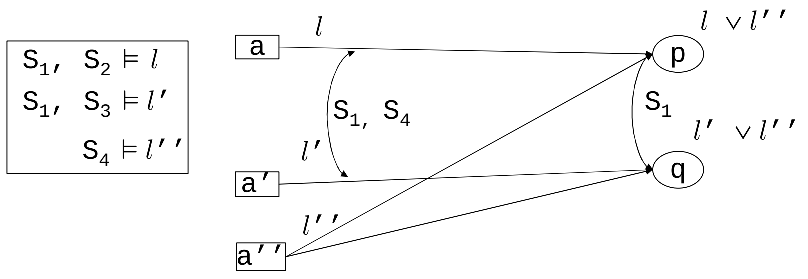

Figure 14: Example of a same-world inconsistent support mutex.

Sum: (h^{LUG}_{sum}) The sum heuristic for the LUG sums the individual levels where each clause in $\kappa(BS_i)$ is fully-supported. Formally:

$$h^{LUG}_{sum}(BS_i) = \sum_{C \in \kappa(BS_i)} \left(\min_{i:FSp^*(C,i)} i \right) \tag{34}$$

For the $CBTC$ problem we need to get the cost of clauses $\neg arm$ and $\neg clog$, as shown in Figure 8 by the circled literals, for $h^{LUG}_{sum}(BS_G)$. The heuristic value is 1 because the level where $\neg arm$ is fully-supported is 1 and for $\neg clog$ is 0.

Level: (h^{LUG}_{level}) The level heuristic is the index of the first level where all the clauses of $\kappa(BS_i)$ are fully-supported. Formally:

$$h^{LUG}_{level}(BS_i) = \min_{i:FSp^*(\kappa(BS_i),i)} i \tag{35}$$

For the $CBTC$ problem we need to find the level of full support for $\neg arm$ and $\neg clog$, as shown in Figure 8 by the literals in squares, for $h^{LUG}_{level}(BS_G)$. The heuristic value is 1 because the level where $\neg arm$ and $\neg clog$ are fully-supported is 1.

Relaxed Plan: (h^{LUG}_{RP}) The relaxed plan heuristic we extract from the LUG is similar to the h^{MG}_{RPU} heuristic. Recall that the h^{MG}_{RPU} heuristic extracts a relaxed plan from each of the multiple planning graphs (one for each possible world) and unions the set of actions chosen at each step in each of the relaxed plans. The LUG relaxed plan heuristic is similar in that it counts actions that are applicable in multiple worlds only once and accounts for actions that are used in subsets of the possible worlds. The advantage is that we find these actions by looking at only one planning graph.

This relaxed plan is representative of a belief space plan because at each level of the graph we ensure that the chosen actions will support the subgoals from all possible worlds.

In many cases the relaxed plan can use one action to supporting subgoals from several possible worlds. This is useful in guiding the search towards plans with lower overall plan length and higher world overlap in achieving the goal from all possible worlds. The relaxed plans extracted from the LUG assume independence between actions because mutex relations are ignored. This property is not as harmful in the contingent planning scenario because action interference can be ignored when actions can be placed in separate branches.

We extract the relaxed plan for a belief state BS_i by starting at level b, where $FSp^*(BS_i, b)$ first holds, by supporting the formula for BS_i with sets of actions at $b-1$ (forming $step_{b-1}$), then support the conjunction of those actions' preconditions at the next lower level of the LUG, and so on. When supporting a formula, it is treated as a CNF. This means that for each clause we find a set of effect relations where (i) each effect gives at least one of the literals in the clause and (ii) the disjunction of labels of literals at b in the clause entails the disjunction of chosen effect relation labels at level $b-1$. The actions in a $step_i$ are determined by the set of effects chosen. This means the possible worlds that can reach the clause are covered by the set of chosen actions. For example, $\{\varphi_1, ..., \varphi_i, ...\}$ at level $b-1$ support a clause with literals $\{l_1, ..., l_j, ...\}$ at level b if:

$$\left(\bigvee_j \ell_b(l_j)\right) \models \left(\bigvee_i \ell_{b-1}(\varphi_i)\right) \tag{36}$$

This is similar to how we find relaxed plans on normal planning graphs, but the differences are in how we determine that a formula is supported by a set of actions and that we prefer supporting with literal persistence before actions. Formally, the value of the relaxed plan heuristic for the LUG is:

$$h_{RP}^{LUG}(BS_i) = \sum_{i=0}^{b-1} \mid step_i \mid \tag{37}$$

A relaxed plan to support BS_G is $DunkP1, DunkP2, Flush$. The first clause, $\neg arm$, is fully-supported through $DunkP1$ and $DunkP2$ because the disjunction of the labels of their conditional effects at level 0 entails the label of $\neg arm$ at level 1. Similarly, $\neg clog$ is fully-supported through $Flush$ because the label of $Flush$'s effect at level 0 entails the label of $\neg clog$ at level 1. Thus $h_{RP}^{LUG}(BS_G) = 3$.

5. Empirical Evaluation

This section presents our implementation of the $\mathcal{C}AltAlt$ and $POND$ planners and the results of our experimentation with the heuristics within them.[14] We also compare with

14. All tests were run in Linux on a Pentium 4 2.66GHz w/ 1GB RAM. Both C*AltAlt* and $POND$ used a weight of five in the, respective, A* and LAO* searches.

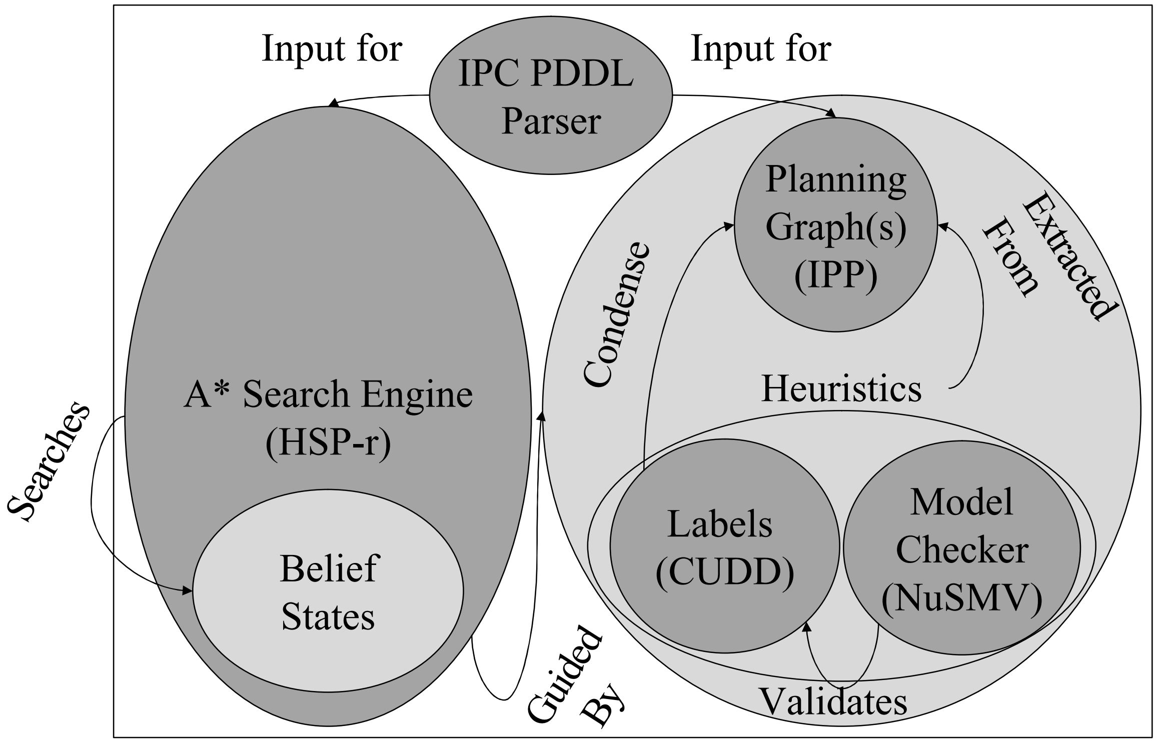

Figure 15: The implementation of C*AltAlt* relies on a regression search engine that searches over belief states. The search engine is guided by heuristics extracted from planning graphs.

the competing approaches (CGP, SGP, GPT 1.40, MBP 0.91, HSCP, and KACMBP) for several domains and problems.[15]

5.1 Implementation

C*AltAlt* The implementation of $\mathcal{C}AltAlt$ uses several off-the-shelf planning software packages. Figure 15 shows a diagram of the system architecture. The components of $\mathcal{C}AltAlt$ are the IPC parser for PDDL 2.1 (slightly extended to allow disjunction) , the HSP-r search engine [Bonet and Geffner, 1999], the IPP planning graph [Koehler *et al.*, 1997], and CUDD [Brace *et al.*, 1990] and NuSMV [Cimatti *et al.*, 2002] to implement the LUG labels. The custom parts of the implementation include the action representation, belief state representation and regression operator, not to mention the heuristic calculation.

15. All domain and problem files for all of the compared planners can be found at http://rakaposhi.eas.asu.edu/belief-search/

Problem	Initial States	Goal Literals	Literals	Causative Actions	Observational Actions	Optimal Parallel	Optimal Serial
$Rovers1$	1	1	66	88	0 {12}	5 {5}	5 {5}
$Rovers2$	2	1	66	88	0 {12}	8 {7}	8 {7}
$Rovers3$	3	1	66	88	0 {12}	10 {?}	10 {8}
$Rovers4$	4	1	66	88	0 {12}	13 {?}	13 {10}
$Rovers5$	16	3	71	97	0 {12}	? {?}	20 {?}
$Rovers6$	12	3	119	217	0 {18}	? {?}	? {?}
$Logistics1$	2	1	29	70	0 {10}	6 {6}	9 {7}
$Logistics2$	4	2	36	106	0 {20}	6 {?}	15 {12}
$Logistics3$	2	1	58	282	0 {21}	8 {?}	11 {8}
$Logistics4$	4	2	68	396	0 {42}	8 {?}	18 {?}
$Logistics5$	8	3	78	510	0 {63}	? {?}	28 {?}
$BT(n)$	n	1	n+1	n	0 {n}	1 {1}	n {n-1}
$BTC(n)$	n	1	n+2	n+1	0 {n}	2n-1 {2}	2n-1 {n-1}
$CubeCorner(n)$	n^3	3	3n	6	0	n-1	3n-3
$CubeFace(n)$	n^3	1	3n	6	0	n	n
$CubeCenter(n)$	n^3	3	3n	6	0	(3n-3)/2	(9n-3)/2
$Ring(n)$	$n3^n$	n	4n	4	0	3n-1	3n-1

Figure 16: Features of test domains and problems - Number of initial states, Number of goal literals, Number of literals, Number of causative actions, Number of Observational Actions, Optimal number of parallel plan steps, Optimal number of serial plan steps. Data for contingent versions of domains is in braces; plan lengths are max contingent branch length.

$POND$ The implementation of $POND$ is very similar to C*AltAlt* aside from the search engine and state representation. $POND$ uses LAO* source code from Eric Hansen to perform the search, and BDDs to represent belief states and actions. Implementation of the heuristics is identical to C*AltAlt*.

5.2 Domains

Figure 16 shows some of the relative features of the different problems we used to evaluate our approach. The table shows the number of initial states, goal literals, literals, actions, and optimal plan lengths. This can be used as a guide to gauge the difficulty of the problems, as well as our performance.

Conformant Problems In addition to the standard domains used in conformant planning–such as Bomb-in-the-Toilet, $Ring$, and $Cube$ variants, we also developed two new domains. We chose these new domains because they demonstrate higher difficulty in the attainment of subgoals, and have many plans of varying length.

The $Rovers$ domain is a conformant adaptation of the analogous domain of the classical planning track of the International Planning Competition [IPC, 2003]. The added uncertainty to the initial state is conditions that rule whether an image objective is visible from various vantage points due to weather as well as the availability of rock and soil samples. The goal is to upload an image of an objective and some rock and soil sample data, thus a conformant plan requires visiting all of the possible vantage points and taking a picture, plus visiting all possible locations of soil and rock samples to draw samples.

The first five $Rovers$ problems have 4 waypoints. Problems one through four have one through four locations, respectively, at which a desired imaging objective is possibly visible (at least one will work, but we don't know which one). Problem 5 adds some rock and soil samples as part of the goal and a couple waypoints where one of each can be obtained (again, we don't know which waypoint will have the right sample). Problem 6 adds two more waypoints, keeps the same goals as Problem 5 and changes the possible locations of the rock and soil samples. In all cases the waypoints are connected in a tree structure, as opposed to completely connected.

The $Logistics$ domain is a conformant adaptation of the classical $Logistics$ domain where trucks and airplanes move packages. The uncertainty is the initial locations of packages. Thus, any actions relating to the movement of packages have a conditional effect that is predicated on the package actually being at a location. In the conformant version, the drivers and pilots cannot sense or communicate a package's actual whereabouts. The problems scale by adding packages and cities.

The $Logistics$ problems consist of one airplane, and cities with an airport, a post office, and a truck. The airplane can travel between airports and the trucks can travel within cities. The first problem has two cities and one package that could start at either post office, and the goal is to get the package to the second city's airport. The second problem adds another package at the same possible starting points and having the same destination. The third problem has three cities with one package that could be at any post office and has to reach the third airport. The fourth problem adds a second package to the third problem with the same starting and ending locations. The fifth problem has three cities, three packages, each at one of two of the three post offices and having to reach different airports.

Contingent Problems For contingent planning we consider several domains from the literature: bomb in the toilet with sensing (BTS), bomb in the toilet with clogging and sensing ($BTCS$), $Medical$ [Weld ***et al.***, 1998], and $Omelette$ [Levesque, 1996]. We also extend the conformant $Logistics$ and $Rovers$ to include observational actions.

The $Rovers$ problem allows for the rover, when it is at a particular waypoint, to sense the availability of image, soil, or rock data at that location. The locations of the collectable data are expressed as one-of constraints, so the rover can deduce the locations of collectable data by failing to sense the other possibilities.

$Logistics$ has observations to determine if a package at a location exists, and the observation is assumed to be made by a driver or pilot at the particular location. Since there are several drivers and a pilot, different agents make the observations. The information gained by the agents is assumed to be automatically communicated to the others, as the planner is the agent that has all the knowledge.[16]

5.3 Conformant Planning

We start by comparing the heuristic approaches within our planners for conformant planning. We continue by describing how our planners, using the best heuristics, compare against other state of the art approaches.

5.3.1 C*AltAlt* AND $POND$

Within C*AltAlt* we show how every heuristic performs for each graph structure, then show how adjusting the computation of mutexes for the LUG can improve performance. Within $POND$ we show how the heuristics perform for the single (SG) and label (LUG) graphs without mutexes, as well as for NG. We abstain from computing mutexes in progression because we build new planning graphs for each search node and we want to keep graph computation time low.

NG heuristics C*AltAlt* performs best, as shown in Figure 17, with the $h_{card'}$ heuristic in the BT and BTC problems (this confirms the results originally seen in [Bertoli *et al.*, 2001a]). However, this heuristic does not perform as well in the $Rovers$ and $Logistics$ problems because the size of a belief state, during planning, does not necessarily indicate that the belief state will be in a good plan. The h_{card} heuristic proves to do worse than the other NG heuristics because it does not take into account enough information. As expected, h_0, giving breadth-first search, does not perform well in a large portion of the problems.

$POND$ (Figures 19 and 20) does slightly better than C*AltAlt* with the NG heuristics because progression admits no inconsistent states. In the tests of Figure 19, the h_0 and $h_{card'}$ heuristics perform similarly in all cases except for BTC. It is interesting to note that the number of nodes expanded by each is almost the same in all problems, showing that $h_{card'}$ is just as uninformed as breadth-first search in progression. However, in the tests of Figure 20, the $h_{card'}$ heuristic does very well by outperforming most of the other heuristics in terms of time and scalability.

SG heuristics For a single planning graph (Figure 17), C*AltAlt* performs best with the h^{SG}_{sum} and h^{SG}_{RP} heuristics, but fails to scale very well on a large portion of the problems. As

16. This problem may be interesting to investigate in a multi-agent planning scenario, assuming no global communication (e.g. no radio dispatcher).

Problem	h_0	h_{card}	$h_{card'}$	h^{SG}_{max}	h^{SG}_{sum}	h^{SG}_{level}	h^{SG}_{RP}
$Rovers1$	2255/5	5580/5	18687/14	66402/5	145604/5	66442/5	543/5
2	49426/8	-	-	9754/8	12483/8	9207/8	78419/8
3	-	-	-	-	42732/10	-	91672/10
4	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
$Logistics1$	1108/9	-	4268/9	737/9	126/9	765/9	198/9
2	-	-	-	-	2282/15	-	7722/15
3	-	-	-	-	1938/14	-	3324/14
4	-	-	-	-	-	-	141094/19
5	-	-	-	-	-	-	-
$BT2$	19/2	19/2	14/2	13/2	16/2	18/2	18/2
10	4837/10	-	56/10	4751/10	4856/10	4842/10	5158/10
20	-	-	418/20	-	-	-	-
30	-	-	1698/30	-	-	-	-
40	-	-	5271/40	-	-	-	-
50	-	-	12859/50	-	-	-	-
60	-	-	26131/60	-	-	-	-
70	-	-	48081/70	-	-	-	-
80	-	-	82250/80	-	-	-	-
$BTC2$	30/3	22/3	16/3	47/3	14/3	14/3	16/3
10	15021/19	-	161/19	14894/19	14850/19	15085/19	15679/19
20	-	-	1052/39	-	-	-	-
30	-	-	3823/59	-	-	-	-
40	-	-	11285/79	-	-	-	-
50	-	-	26514/99	-	-	-	-
60	-	-	55687/119	-	-	-	-
70	-	-	125594/140	-	-	-	-

Figure 17: Results for C*AltAlt* using all NG, and SG heuristics for conformant $Rovers$, $Logistics$, BT, and BTC. The data is Total Time / # Expanded Nodes, “-” indicates no solution.

Problem	h^{MG}_{max}	h^{MG}_{sum}	h^{MG}_{level}	h^{MG}_{RPM}	h^{MG}_{RPU}	$h^{LUG(FX)}_{max}$	$h^{LUG(FX)}_{sum}$	$h^{LUG(FX)}_{level}$	$h^{LUG(FX)}_{RP}$
$Rovers1$	62266/5	142516/5	62398/5	542/5	185/5	15809/5	16582/5	15153/5	15164/5
2	91815/8	90256/8	96642/8	8327/8	29285/9	33991/8	40348/8	32599/8	32969/8
3	82695/10	91747/10	92890/10	20162/10	2244/11	58146/10	26596/10	16924/10	16668/10
4	-	-	-	61521/16	3285/15	-	-	32490/13	31584/13
5	-	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-	-
$Logistics1$	1013/9	201/9	1098/9	183/9	1109/9	2434/9	1395/9	1603/9	1340/9
2	19051/15	15766/15	23648/15	15491/15	69818/19	-	59379/15	39656/15	18535/15
3	-	6231/14	-	70882/14	-	-	22330/14	32142/14	16458/15
4	-	-	-	-	-	-	-	-	178068/19
5	-	-	-	-	-	-	-	-	-
$BT2$	18/2	17/2	18/2	20/2	21/2	50/2	11/2	11/2	12/2
10	5366/10	4808/10	6062/10	8988/10	342/10	4846/10	4813/10	4791/10	71/10
20	-	-	-	-	2299/20	-	-	-	569/20
30	-	-	-	-	9116/30	-	-	-	2517/30
40	-	-	-	-	44741/40	-	-	-	7734/40
50	-	-	-	-	-	-	-	-	18389/50
60	-	-	-	-	-	-	-	-	37820/60
70	-	-	-	-	-	-	-	-	70538/70
80	-	-	-	-	-	-	-	-	188603/80
$BTC2$	15/3	14/3	15/3	33/3	23/3	231/3	18/3	19/3	18/3
10	15863/19	14874/19	17526/19	41805/19	614/19	16811/19	16633/19	16818/19	1470/19
20	-	-	-	-	2652/39	-	-	-	51969/39
30	-	-	-	-	9352/59	-	-	-	484878/59
40	-	-	-	-	51859/79	-	-	-	-
50	-	-	-	-	-	-	-	-	-
60	-	-	-	-	-	-	-	-	-
70	-	-	-	-	-	-	-	-	-

Figure 18: Results for C*AltAlt* using all MG, and LUG heuristics for conformant $Rovers$, $Logistics$, BT, and BTC. The data is Total Time / # Expanded Nodes, "-" indicates no solution. Here the LUG heuristics use all cross-world mutexes (the FX scheme).

in classical planning [Nguyen *et al.*, 2002], sum and relaxed plan heuristics, despite being inadmissible, tend to generate solutions much more quickly than the more conservative max and level heuristics.

In $POND$, Figures 19 and 20, the SG heuristics performed similarly, providing search guidance comparable to breath-first search (as evidenced by the number of expanded nodes), but with the added cost of computation (as evidenced by the total time).

MG heuristics As with the SG heuristics, Figure 18, C*AltAlt* using MG does best with the sum and relaxed plan heuristics. An interesting point to note on the BT and BTC problems is that the h^{MG}_{RPU} significantly outperforms the others because it accounts for different actions being used for each possible world. The relaxed plan heuristics tend to expand fewer search nodes than the other heuristics, but in some cases their total planning time is longer because they take longer to compute.

$POND$ was not evaluated with the MG heuristics because of the prohibitive cost of building multiple planning graphs for each search node.

LUG heuristics In C*AltAlt*, Figure 18, the h^{LUG}_{RP} heuristic outperforms all other heuristics on all problems. The relaxed plan works well because it helps give better discrimination between search choices, where other heuristics may give many similar evaluations for search nodes. Here the mutex scheme for the LUG is the (FX) scheme using all possible cross and same world mutexes. Naturally, this increases the LUG construction time as problems become more complex. Shortly, we will consider the relaxations to computing mutexes to improve graph computation time and scalability.

In $POND$, Figures 19 and 20, the LUG heuristics also do the best. The relaxed plan heuristic tends to scale best because it expands fewer search nodes. The use of helpful actions [Hoffmann and Nebel, 2001] in h^{LUG}_{RP-ha} is also useful in cutting down time in the $Logistics$, $Rovers$, and $CubeCenter$ problems, but for other problems it does not help as much.

LUG Mutexes Since the LUG is used for heuristic guidance only and the number of possible mutexes we can find is quite large, we can use several schemes to relax the complexity of the mutex computations. The schemes combine different types of mutexes with types of cross world checking. The mutex types are: (NX) computing no mutexes, (StX) computing only interference mutexes, (DyX) computing (StX) plus inconsistent support, and competing needs mutexes, and (FX) computing (DyX) plus induced mutexes. The cross world checking reduction schemes are: (SX) computing mutexes across same-worlds (mentioned previously in this section) and (IX) computing mutexes across pairs of worlds in the intersection (conjunction) of element labels.

Figure 21 shows that within C*AltAlt*, using the relaxed plan heuristic and changing the way we compute mutexes on the LUG can drastically alter performance. Often, the possible number of cross world mutexes are so numerous that building the LUG takes too much time. So, we evaluated h^{LUG}_{RP} when the LUG is built (a) considering all cross world relations, for the schemes (NX), (StX), (DyX), and (FX); and (b) same world relations for the schemes (DyX-SX) and (FX-SX), and (c) cross world relations for all possible worlds pairs in the intersection of element's labels (DyX-IX) and (FX-IX).

The results show that simpler problems like BT and BTC do not benefit as much from advanced computation of mutexes beyond static interference. However, for the $Rovers$ and $Logistics$ problems, advanced mutexes play a larger role. Mainly, interference, competing needs, and inconsistent support mutexes are important. The competing needs and

Problem	h_0	$h_{card'}$	h^{SG}_{max}	h^{SG}_{sum}	h^{SG}_{level}	h^{SG}_{RP}	h^{LUG}_{max}	h^{LUG}_{sum}	h^{LUG}_{level}	h^{LUG}_{RP}	h^{LUG}_{RP-ha}
$Rovers$1	230/26	220/26	470/16	700/26	470/16	710/26	1580/26	1580/26	330/5	360/5	230/5
2	1970/187	1970/187	4430/137	5160/187	4430/137	5260/187	15170/187	15050/187	2830/30	3500/34	1790/21
3	11340/897	11460/897	27720/765	28190/897	28220/765	29040/897	88760/897	-	7640/65	8220/62	7570/62
4	75840/4725	76230/4725	-	-	-	-	-	-	30620/207	28670/192	27650/191
5	-	-	-	-	-	-	-	-	-	82240/235	65510/172
6	-	-	-	-	-	-	-	-	-	-	-
$Logistics$1	280/95	280/95	880/105	820/95	870/105	940/105	990/40	990/40	450/15	420/12	330/11
2	21090/2314	21970/2560	-	-	-	-	22950/304	18680/223	8250/112	2040/18	1590/17
3	60840/2208	37160/1361	-	-	-	-	71880/511	71700/511	31400/201	7510/36	3770/21
4	-	-	-	-	-	-	-	-	-	34830/117	13400/49
5	-	-	-	-	-	-	-	-	-	-	50650/62
BT2	0/2	0/2	0/2	0/2	0/2	10/2	0/2	0/2	0/2	0/2	0/2
10	20/10	20/10	80/10	90/10	90/10	90/10	130/10	130/10	130/10	120/10	130/10
20	320/20	320/20	1010/20	1010/20	980/20	1020/20	1470/20	1460/20	1470/20	1530/20	1550/20
30	1970/30	1960/30	4530/30	4540/30	4540/30	4670/30	6800/30	6870/30	6820/30	7060/30	7170/30
40	6550/40	6500/40	13660/40	13660/40	13710/40	13870/40	20180/40	20340/40	20380/40	20860/40	21040/40
50	18100/50	17920/50	34150/50	34130/50	34120/50	35000/50	51910/50	52350/50	51960/50	53810/50	54160/50
60	40380/60	43600/60	-	88360/60	87780/60	88020/60	119740/60	120110/60	-	-	122110/60
70	95610/70	96150/70	-	-	-	-	-	-	-	-	-
80	261260/80	264300/80	-	-	-	-	-	-	-	-	-
BTC2	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3
10	2340/19	2420/19	5230/19	5200/19	5170/19	5500/19	7300/19	7340/19	7390/19	230/19	230/19
20	-	440/39	-	-	-	-	-	-	-	2830/39	2840/39
30	-	2220/59	-	-	-	-	-	-	-	12060/59	12180/59
40	-	7400/79	-	-	-	-	-	-	-	35950/79	36750/79
50	-	18510/99	-	-	-	-	-	-	-	91170/99	95130/99
60	-	40140/119	-	-	-	-	-	-	-	-	-
70	-	91460/139	-	-	-	-	-	-	-	-	-

Figure 19: Results for $POND$ using all NG, SG, and LUG heuristics for conformant $Rovers$, $Logistics$, BT, and BTC. The data is Total Time / # Expanded Nodes, "-" indicates no solution.

Problem	h_0	$h_{card'}$	h^{SG}_{max}	h^{SG}_{sum}	h^{SG}_{level}	h^{SG}_{RP}	h^{LUG}_{max}	h^{LUG}_{sum}	h^{LUG}_{level}	h^{LUG}_{RP}	h^{LUG}_{RP-ha}
CubeCnr3	40/85	10/25	230/94	230/85	250/94	270/94	50/6	50/6	340/94	70/6	60/6
4	640/661	120/299	2250/720	2260/661	2260/720	2410/720	170/9	180/9	1740/175	200/9	210/9
5	5220/2540	540/787	12710/2481	12760/2540	13090/2481	13790/2481	430/12	440/12	8210/444	530/12	540/12
6	39720/7979	2000/1815	-	-	-	-	950/15	940/15	33300/1016	1160/15	1210/15
7	184360/18721	6260/3526	-	-	-	-	1830/18	1830/18	-	2240/18	2380/18
8	715640/40078	20620/6667	-	-	-	-	3290/21	3260/21	-	3980/21	4310/21
9	-	43810/9966	-	-	-	-	5560/24	5540/24	-	6750/24	7420/24
10	-	95670/15181	-	-	-	-	9280/27	9300/27	-	11280/27	12380/27
11	-	201960/22550	-	-	-	-	15830/30	15830/30	-	19090/30	20990/30
12	-	359680/30180	-	-	-	-	26500/33	26440/33	-	31030/33	34070/33
13	-	-	-	-	-	-	41970/36	41960/36	-	48490/36	52910/36
14	-	-	-	-	-	-	64950/39	65120/39	-	72870/39	79530/39
15	-	-	-	-	-	-	96610/42	96650/42	-	108610/42	118250/42
16	-	-	-	-	-	-	144270/45	144050/45	-	162060/45	176400/45
17	-	-	-	-	-	-	217990/48	219840/48	-	242190/48	260460/48
18	-	-	-	-	-	-	344380/51	341580/51	-	373370/51	401070/51
19	-	-	-	-	-	-	-	535870/54	-	585440/54	-
CubeCtr3	60/196	10/40	310/198	300/196	300/198	330/198	360/193	350/154	310/196	340/165	180/41
4	480/423	20/24	1520/337	1710/423	1500/337	1600/255	1790/423	1810/423	1750/423	510/50	120/9
5	5910/3284	430/661	14730/3356	14610/3284	14790/3356	15610/3356	19820/3301	17000/2132	18350/3300	15240/1567	15180/1419
6	40280/8180	730/762	-	-	-	-	-	-	-	35430/1630	16910/604
7	211380/21936	5580/3292	-	-	-	-	-	-	-	-	-
8	-	8510/3883	-	-	-	-	-	-	-	-	-
9	-	47730/10570	-	-	-	-	-	-	-	-	-
10	-	54990/10780	-	-	-	-	-	-	-	-	-
11	-	215520/22869	-	-	-	-	-	-	-	-	-
12	-	236960/22385	-	-	-	-	-	-	-	-	-
CubeFace3	10/17	10/9	50/11	80/17	60/11	70/11	120/16	120/16	20/2	20/2	0/2
4	170/88	20/13	480/66	590/88	480/66	560/66	620/45	610/45	80/3	80/3	20/3
5	1080/219	60/17	2510/224	2430/219	2500/224	2800/224	2780/109	2760/109	190/4	210/4	70/4
6	5460/502	150/21	10120/513	9510/502	10130/513	10810/513	10050/235	9920/235	430/5	480/5	140/5
7	19310/893	330/25	42490/1345	29440/893	42460/1345	45090/1345	31150/443	30710/443	850/6	940/6	300/6
8	65400/1678	690/29	-	91050/1678	-	-	86810/782	85360/782	1600/7	1710/7	0570/7
9	202670/3038	1410/33	-	-	-	-	-	-	2920/8	3080/8	1000/8
10	-	2990/37	-	-	-	-	-	-	5400/9	5760/9	1830/9
11	-	7370/41	-	-	-	-	-	-	10730/10	11100/10	3400/10
12	-	15970/45	-	-	-	-	-	-	19490/11	19870/11	6100/11
13	-	29780/49	-	-	-	-	-	-	33310/12	33710/12	10500/12
14	-	49990/53	-	-	-	-	-	-	53760/13	54830/13	16640/13
15	-	81440/57	-	-	-	-	-	-	80870/14	81720/14	25170/14
16	-	127450/61	-	-	-	-	-	-	122260/15	123350/15	37400/15
17	-	198680/65	-	-	-	-	-	-	184410/16	186190/16	55300/16
18	-	304720/69	-	-	-	-	-	-	275860/17	276490/17	80950/17
19	-	430140/73	-	-	-	-	-	-	411220/18	411520/18	118710/18
20	-	-	-	-	-	-	-	-	-	-	172970/19
Ring2	0/13	10/7	10/12	30/13	20/12	10/12	30/13	30/13	20/8	30/7	20/7
3	100/62	30/11	200/61	190/62	180/61	200/61	0390/62	410/62	320/36	530/51	530/51
4	1570/239	130/15	2230/249	2210/239	2220/249	2290/249	4730/239	4750/239	4680/224	6490/221	6460/221
5	24540/1017	840/19	30200/1004	30170/1017	30190/1004	30240/1004	51940/1017	52900/1017	49170/792	70330/938	70620/938
6	554350/4066	7830/23	-	-	-	-	-	-	-	-	-
7	-	55920/27	-	-	-	-	-	-	-	-	-
8	-	292230/31	-	-	-	-	-	-	-	-	-

Figure 20: Results for $POND$ using all NG, SG, and LUG heuristics for conformant $CubeCorner$, $CubeCenter$, $CubeFace$, and $Ring$. The data is Total Time / # Expanded Nodes, “-” indicates no solution.

Problem	$h_{RP}^{LUG(NX)}$	$h_{RP}^{LUG(StX)}$	$h_{RP}^{LUG(DyX)}$	$h_{RP}^{LUG(FX)}$	$h_{RP}^{LUG(DyX-SX)}$	$h_{RP}^{LUG(DyX-IX)}$	$h_{RP}^{LUG(FX-SX)}$	$h_{RP}^{LUG(FX-IX)}$
$Rovers1$	13/1112/51	19/1119/51	15453/89/6	15077/87/6	15983/87/6	15457/87/6	15098/86/6	15094/85/6
2	20/904/41	16/903/41	13431/138/8	32822/147/8	10318/139/8	10625/134/8	10523/138/8	14550/138/8
3	13/8704/384	17/8972/384	17545/185/10	16481/187/10	10643/185/10	11098/209/10	10700/191/10	11023/184/10
4	-	-	32645/441/14	31293/291/14	14988/291/14	16772/291/14	14726/290/14	16907/290/14
5	-	-	698575/3569/45	-	61373/3497/45	379230/3457/45	60985/3388/45	378869/3427/45
6	-	-	-	-	217507/3544/37	565013/3504/37	225213/3408/37	588336/3512/37
$Logistics1$	5/868/81	10/868/81	1250/117/9	1242/98/9	791/116/9	797/117/9	796/115/9	808/115/9
2	10/63699/1433	88/78448/1433	16394/622/15	18114/421/15	2506/356/15	7087/428/15	2499/352/15	6968/401/15
3	-	-	17196/1075/15	16085/373/15	10407/403/15	10399/408/15	10214/387/15	10441/418/15
4	-	-	136702/1035/19	176995/1073/19	24214/648/19	71964/871/19	23792/642/19	71099/858/19
5	-	-	-	-	52036/2690/41	328114/4668/52	52109/2672/41	324508/4194/52
$BT2$	1/34/2	0/13/2	0/13/2	0/12/2	0/16/2	0/15/2	0/25/2	0/13/2
10	4/72/10	4/56/10	13/57/10	13/58/10	12/59/10	14/59/10	13/59/10	14/56/10
20	19/452/20	22/448/20	120/453/20	120/449/20	102/450/20	139/454/20	105/444/20	137/454/20
30	62/1999/30	59/1981/30	514/1999/30	509/2008/30	421/1994/30	600/2007/30	413/1986/30	596/2002/30
40	130/6130/40	132/6170/40	1534/6432/40	1517/6217/40	1217/6326/40	1822/6163/40	1196/6113/40	1797/6127/40
50	248/14641/50	255/14760/50	3730/14711/50	3626/14763/50	2866/14707/50	4480/14676/50	2905/14867/50	4392/14683/50
60	430/30140/60	440/29891/60	7645/30127/60	7656/30164/60	5966/30017/60	9552/30337/60	5933/30116/60	9234/29986/60
70	680/55202/70	693/55372/70	15019/55417/70	14636/55902/70	11967/55723/70	18475/55572/70	11558/55280/70	18081/55403/70
80	1143/135760/80	1253/140716/80	26478/132603/80	26368/162235/80	21506/136149/80	32221/105654/80	21053/139079/80	32693/109508/80
$BTC2$	0/62/3	1/16/3	0/15/3	4/14/3	0/16/3	1/14/3	1/13/3	2/14/3
10	4/93/19	4/77/19	14/78/19	1388/82/19	13/76/19	16/75/19	14/75/19	440/81/19
20	21/546/39	32/545/39	139/553/39	51412/557/39	105/546/39	140/549/39	110/555/39	19447/568/39
30	58/2311/59	61/2293/59	543/2288/59	482578/2300/59	427/2294/59	606/2300/59	444/2287/59	199601/2401/59
40	133/6889/79	149/6879/79	1564/6829/79	-	1211/6798/79	1824/6816/79	1253/6830/79	1068019/???/79
50	260/15942/99	261/16452/99	-	-	2890/16184/99	4412/16414/99	2926/16028/99	-
60	435/32201/119	443/32923/119	-	-	6045/32348/119	9492/32350/119	6150/32876/119	-
70	742/62192/139	745/61827/139	-	-	-	-	-	-

Figure 21: Results for C*AltAlt* using h_{RP}^{LUG} heuristic with different mutex schemes for conformant $Rovers$, $Logistics$, BT, and BTC. The data is Graph Construction Time / All Other Non Graph Construction Time / # Expanded Nodes, "-" indicates no solution.

inconsistent support mutexes seem to have a large impact on the informedness of the guidance given by the LUG, as scalability improves most here. Induced mutexes don't improve search time much, and only add to graph computation time. Reducing cross world mutex checking also helps quite a bit. It seems that only checking same world mutexes is sufficient to solve large problems. Interestingly, the MG graphs compute same-world interference, competing needs, and inconsistent support mutexes within each graph, equating to the same scenario as (DyX-SX), however, the LUG provides a much faster construction time, evidenced by the LUG's ability to out-scale MG.

Comparison of Graph Types All of the heuristics from the single graph fail to account for different possible worlds supporting the same literals and reduce to BFS in the BT and BTC problems. However, single graph heuristics can leverage a bit more structure in the $Rovers$ and $Logistics$ problems because each possible world has several actions that need to be performed. The multiple graph heuristics tend to account better for the multiple possible worlds of the problems and give better estimates. However, the downfall of multiple graphs is the large construction time and exponential increase of the number of graphs with the number of uncertain projected belief state literals. The labelled uncertainty graph takes the best of the single graph and multiple graphs by having a fast computation time, compact structure, and a wealth of information to compute informed heuristics.

In summary, we have learned three points:

- The labelled uncertainty graph with the relaxed plan heuristic is the most cost-effective and informed heuristic that we have presented.
- Reachability heuristics are needed for conformant planning domains adapted from IPC classical planning domains.
- Same world mutexes from the (DyX-SX) mutex scheme for the LUG provide the best cost benefit ratio for regression search.

5.3.2 COMPARISON WITH OTHER CONFORMANT PLANNERS

Although this work is aimed at giving a general comparison of heuristics for belief space planning, we also present a comparison of the best heuristics within $\mathcal{C}AltAlt$ and $POND$ to some of the other leading approaches to conformant planning. Note, since each approach uses a different planning representation (BDDs, GraphPlan, or explicit state space), not all of which even use heuristics, it is hard to get a standardized comparison of heuristic effectiveness. Furthermore, not all of the planners use PDDL-like syntax; MBP, HSCP, and KACMBP use AR encodings which may give them an advantage in reducing the number

Problem	C*AltAlt* $h_{RP}^{LUG(DyX-SX)}$	POND h_{RP-ha}^{LUG}	MBP	KACMBP	HSCP	GPT	CGP	SGP
$Rovers$1	16070/5	230/5	66/5	9293/5	-	3139/5	70/5	70/5
2	10457/8	1790/8	141/8	9289/15	-	4365/8	180/8	30/8
3	10828/10	7570/15	484/10	9293/16	-	5842/10	460/10	1750/10
4	15279/13	27650/19	3252/13	9371/18	-	7393/13	1860/13	-
5	64870/29	65510/33	-	39773/40	-	399525/20	-	-
6	221051/25	-	727/32	-	-	-	-	-
$Logistics$1	907/9	330/11	37/9	127/12	352/9	916/9	60/6	70/6
2	2862/15	1590/17	486/24	451/19	-	1297/15	290/6	510/6
3	10810/15	3770/14	408/14	1578/18	-	1711/11	400/8	4620/8
4	24862/19	13400/21	2881/27	8865/22	-	9828/18	1170/8	447470/8
5	54726/34	50650/30	-	226986/42	-	543865/28	-	-
BT2	16/2	0/2	6/2	10/2	8/2	487/2	20/1	0/1
10	71/10	130/10	119/10	16/10	6/10	627/10	520/1	30/1
20	552/20	1550/20	-	84/20	23/20	472174/20	3200/1	290/1
30	2415/30	7170/30	-	244/30	47/30	-	10330/1	1170/1
40	7543/40	21040/40	-	533/40	80/40	-	24630/1	3320/1
50	17573/50	54160/50	-	1090/50	148/50	-	49329/1	7550/1
60	35983/60	122110/60	-	2123/60	340/60	-	87970/1	83494/1
70	67690/70	-	-	3529/70	-	-	145270/1	114340/1
80	157655/80	-	-	-	-	-	-	-
BTC2	16/3	0/3	8/3	18/3	2/3	465/3	0/3	0/3
10	89/19	230/19	504/19	45/19	25/19	715/19	39370/19	-
20	651/39	2840/39	98/39	211/39	98/39	-	-	-
30	2721/59	12180/59	268/59	635/59	293/59	-	-	-
40	8009/79	36750/79	615/79	1498/79	674/79	-	-	-
50	19074/99	95130/99	1287/99	10821/99	1352/99	-	-	-
60	38393/119	-	2223/119	5506/119	5100/119	-	-	-
70	-	-	3625/139	9334/139	-	-	-	-

Figure 22: Results for C*AltAlt* using $h_{RP}^{LUG(DyX-SX)}$, $POND$ using h_{RP-ha}^{LUG}, MBP, KACMBP, HSCP, GPT, CGP, and SGP for conformant $Rovers$, $Logistics$, BT, and BTC. The data is Total Time / # Plan Steps, "-" indicates no solution.

of literals and actions.[17] Nevertheless, Figure 22 compares MBP, KACMBP, HSCP, GPT, CGP, and SGP with $h_{RP}^{LUG(DyX-SX)}$ in C*AltAlt* and h_{RP-ha}^{LUG} in $POND$ with respect to run time and plan length.

An observation independent of the planning substrate is the optimality of plans. Optimality can be ensured by using admissible heuristics, but of the heuristic approaches that are inadmissible it is interesting to note that in many cases MBP, HSCP, and KACMBP tend to generate plans that are *much longer* than plans generated by $\mathcal{C}AltAlt$ using $h_{RP}^{LUG(DyX-SX)}$ in the $Rovers$ and $Logistics$ domains. Furthermore, the optimal approaches (CGP, SGP, and GPT) cannot scale as well, despite their good solutions.

For $Rovers$, $h_{RP}^{LUG(DyX-SX)}$ in C*AltAlt* provides the best guidance by outperforming all other planners on most of the instances. $POND$ outperforms C*AltAlt* in the $Logistics$ problems, and is competitive with the other planners. GPT finds optimal serial plans but is not as effective when the size of the search space increases. CGP has trouble constructing its planning graphs as the conformant depth of the goal increases. The LUG in $\mathcal{C}AltAlt$ can handle large domains better than CGP's planning graphs, and thus $h_{RP}^{LUG(DyX-SX)}$ scales much better. MBP does outperform C*AltAlt* using $h_{RP}^{LUG(DyX-SX)}$ in some cases on $Rovers$ problems, but at the cost of generating sub-optimal plans (most likely a by-product of its depth first search).[18] $Logistics$ is a more fertile domain for comparisons. The lesson to be learned is that HSCP's cardinality heuristic, similar to $h_{card'}$, does not scale well. Yet HSCP does better than C*AltAlt* with $h_{card'}$, indicating that the planning substrate, as opposed to the heuristic, may be responsible for the performance. The BT and BTC domains show that $\mathcal{C}AltAlt$ is competitive with CGP and GPT, but is dominated by HSCP and KACMBP with respect to handling common structure in problems.

In summary, we have shown that C*AltAlt* with the $h_{RP}^{LUG(DyX-SX)}$ heuristic and $POND$ with the similar h_{RP-ha}^{LUG} heuristic provide superior search guidance for conformant planning problems adapted from the IPC. The same heuristics prove to also give good guidance for traditional conformant planning problems.

5.4 Contingent Planning

In contingent planning we first discuss the effectiveness of our heuristics in $POND$, and then compare to competing planners.

17. We gave the MBP planners the same grounded and filtered action descriptions that we used in C*AltAlt* and $POND$. We also tried, but do not report results, giving the MBP planners the full set of ground actions without filtering irrelevant actions. It appears that the MBP planners do not use any sort of action pre-processing because performance was much worse with the full grounded set of actions.

18. Surprisingly, MBP (used here in depth first progression search with a cardinality heuristic) performs well, beating out KACMBP and HSCP on almost all the Logistics and Rovers domains. This is strange because KACMBP and HSCP are supposed to be improvements upon the original MBP.

5.4.1 $POND$

Figure 23 shows that within $POND$, the LUG heuristics generate better plans than blind search and the cardinality heuristic because they do not model any notion of reachability for belief space. This is evident in the $Logistics$ and $Rovers$ problems (where reachability is important) through better plans and scalability. We refer the reader to Appendix C for a sample plan found by $POND$ for the most difficult $Rovers$ problem. The LUG is somewhat expensive to build for every search node, so we tried helpful actions [Hoffmann and Nebel, 2001]. Helpful actions are actions that have the same effects as actions in the first step of a relaxed plan; the search's branching factor is drastically decreased by considering helpful actions, at the expense of an incomplete search. Helpful actions improved $POND$'s scalability in the Logistics and Rovers domains because the LUG is already giving good guidance to the planner, but we avoid wasting time generating un-helpful belief states with un-helpful actions.

5.4.2 COMPARISON WITH OTHER CONTINGENT PLANNERS

Figure 24 shows the results for testing the contingent versions of the domains on $POND$, MBP, GPT, and SGP.[19] The major point to notice is that MBP tends to generate solutions much faster but of very poor quality, and GPT and SGP generate better solutions but very slowly. The $POND$ planner is very similar to MBP in that it uses progression search. However, $POND$ uses an LAO* search, whereas the MBP binary we used uses a depth first And-Or search. The DFS used by MBP contributes to highly sub-optimal max length branches (as much as **25 times longer** than with $POND$ with h^{LUG}_{RP-ha}), but allows for efficient plan generation times. For instance, the plans generated by DFS in MBP for the Rovers domain have the rover navigating back and forth between locations several times before doing anything useful; this is not a situation beneficial for actual mission use. A reason why the quality of plans generated by MBP has not previously come into question is that MBP has only been evaluated on problems that do not have a variety of feasible solutions of different lengths.

5.5 Comprehensive Summary of Empirical Results

- Planning graph heuristics for belief space search help control conformant and contingent plan length because, as opposed to cardinality, the heuristics model desirable plan quality metrics.

- Planning graph heuristics for belief space search scale better than planning graph search and admissible heuristic search techniques.

19. There were a few differences in the problem encodings related to observation preconditions. The MBP planner does not allow preconditions to observations, so the planner could observe variables at any time.

Problem	h_0	$h_{card'}$	h^{SG}_{max}	h^{SG}_{sum}	h^{SG}_{level}	h^{SG}_{RP}	h^{LUG}_{max}	h^{LUG}_{sum}	h^{LUG}_{level}	h^{LUG}_{RP}	h^{LUG}_{RP-ha}
$Rovers$1	220/26	220/26	470/16	690/26	450/16	710/26	1570/26	1590/26	340/5	350/5	230/5
2	870/108	300/38	2140/58	2960/108	2120/58	2700/95	7210/95	7230/95	900/12	850/11	620/11
3	1110/144	410/54	2890/75	4180/144	2900/75	2700/86	7580/86	7570/86	1340/16	1310/15	990/15
4	1660/205	500/67	4700/131	6340/205	4680/131	4900/158	11920/131	11950/131	2510/27	1920/20	1490/20
5	-	-	-	-	-	-	-	-	-	32750/208	21510/182
6	-	-	-	-	-	-	-	-	-	88890/210	48730/156
$Logistics$1	190/69	100/35	500/33	630/69	500/33	540/33	1330/56	1310/56	290/10	320/10	210/10
2	10620/1616	2360/476	-	-	-	-	29600/811	32880/417	2440/45	2370/36	1930/36
3	14490/657	3250/189	-	-	-	-	-	-	3210/18	3390/19	2100/19
4	-	103440/2625	-	-	-	-	-	-	-	23620/83	11300/54
5	-	-	-	-	-	-	-	-	-	-	-
BT2	0/3	0/3	0/2	0/3	0/2	0/3	0/2	0/2	0/3	0/3	0/3
10	10/19	20/19	130/19	120/19	130/19	130/19	190/19	190/19	180/19	190/19	200/19
20	220/39	210/39	1250/39	1240/39	1240/39	1280/39	1950/39	1900/39	1930/39	2030/39	2090/39
30	1200/59	1200/59	5200/59	5200/59	5200/59	5300/59	8370/59	8330/59	8370/59	8930/59	9060/59
40	3670/79	3830/79	14790/79	14740/79	14780/79	15150/79	24020/79	23890/79	24000/79	26180/79	26420/79
50	10990/99	11810/99	36380/99	37960/99	36150/99	37290/99	61160/99	59910/99	60570/99	65170/99	66320/99
60	23170/119	23400/119	-	-	-	-	-	-	-	-	-
70	54050/139	53750/139	-	-	-	-	-	-	-	-	-
80	150420/159	153360/159	-	-	-	-	-	-	-	-	-
BTC2	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	10/3	0/3
10	20/19	20/19	160/19	160/19	160/19	170/19	540/29	540/29	240/19	310/19	310/19
20	230/39	250/39	1570/39	1570/39	1560/39	1640/39	2720/39	2730/39	2700/39	3380/39	3400/39
30	1250/59	1260/59	6130/59	6150/59	6100/59	6330/59	11240/59	11290/59	11190/59	13990/59	14100/59
40	4110/79	4100/79	17980/79	18440/79	18080/79	18290/79	33690/79	33530/79	33770/79	41820/79	42580/79
50	10620/99	12490/99	43410/99	43330/99	44140/99	43550/99	83560/99	81840/99	83160/99	103900/99	-
60	22160/119	23680/119	-	-	-	-	-	-	-	-	-
70	46360/139	48240/139	-	-	-	-	-	-	-	-	-
$Omelette$1	10/6	10/6	60/5	70/6	70/5	80/5	-	-	100/5	100/5	130/8
2	10/12	30/21	200/10	160/12	210/10	220/10	-	-	320/14	310/11	-
3	80/50	70/63	480/28	770/50	480/28	510/28	-	-	1570/48	1240/31	-
$Medical$1	0/1	0/1	0/1	10/1	0/1	0/1	80/1	80/1	80/1	70/1	80/1
2	0/5	0/4	20/14	20/14	20/14	10/14	2000/14	2010/14	1560/4	2630/12	-
3	0/7	0/7	30/22	30/22	30/22	40/20	4830/25	4870/25	3290/7	5570/19	-
4	10/9	0/10	80/54	60/54	60/54	70/25	10570/52	10570/52	5860/10	8210/22	-
5	0/11	0/12	80/47	90/47	80/47	70/27	13110/54	13230/54	8460/12	13290/27	-

Figure 23: Results for $POND$ using all NG, SG, and LUG heuristics for contingent $Rovers$, $Logistics$, BTS, $BTCS$, $Omelette$, and $Medical$. The data is Total Time / # Expanded Nodes, “-” indicates no solution.

Problem	POND h^{LUG}_{RP-ha}	MBP	GPT	SGP
$Rovers$1	230/5	3328/11	3148/5	70/5
2	620/7	7066/57	5334/7	760/7
3	990/8	4818/61	7434/8	-
4	1490/10	4939/57	11430/10	-
5	21510/19	1389/81	-	-
6	48730/23	3145/158	-	-
$Logistics$1	210/7	31/17	1023/7	5490/6
2	1930/12	-	5348/12	-
3	2100/9	2120/45	2010/8	-
4	11300/17	1086/650	-	-
5	-	5768/1977	-	-
BT2	0/2	8/3	510/2	0/1
10	200/10	12/19	155314/10	70/1
20	2090/20	78/39	-	950/1
30	9060/30	305/59	-	4470/1
40	26420/40	887/79	-	13420/1
50	66320/50	2350/99	-	32160/1
60	-	4479/119	-	90407/1
70	-	-	-	120010/1
80	-	-	-	-
BTC2	0/2	10/3	529/2	10/2
10	310/10	53/19	213277/10	-
20	3400/20	518/39	-	-
30	14100/30	2551/59	-	-
40	42580/40	8475/79	-	-
50	-	20684/99	-	-
60	-	47375/119	-	-
70	-	-	-	-
$Omelette$1	130	-	250	-
2	-	-	4297	-
3	-	-	12866	-
$Medical$1	80/1	12/3	506/1	0/1
2	-	12/3	514/3	20/3
3	-	6/3	541/5	70/3
4	-	12/4	531/5	630/3
5	-	16/4	547/5	4550/3

Figure 24: Results for $POND$ using h^{LUG}_{RP-ha}, MBP, GPT, and SGP for contingent $Rovers$, $Logistics$, BT, BTC, $Omelette$, and $Medical$. The data is Total Time / # Max possible steps in a execution, "-" indicates no solution.

- Of the planning graph heuristics presented, relaxed plans that take into account the overlap of individual plans between states of the source and destination belief states are the most informed.
- The LUG is an effective planning graph data structure for both regression and progression search heuristics.
- In regression search, planning graphs that maintain only same-world mutexes provide the best trade-off between graph construction cost and heuristic informedness.
- The LUG heuristics help our contingent planner, $POND$, to scale up and well, despite the fact that the heuristic computation phase does not model observation actions.

6. Related Work

The recent interest in conformant and contingent planning can be traced to CGP [Smith and Weld, 1998], a conformant version of GraphPlan [Blum and Furst, 1995], and SGP [Weld *et al.*, 1998], the analogous contingent version of GraphPlan. Here the graph search is conducted on several planning graphs, each constructed from one of the possible initial states. More recent work on C-plan [Castellini *et al.*, 2001] and Frag-Plan [Kurien *et al.*, 2002] generalize the CGP approach by ordering the searches in the different worlds such that the plan for the hardest to satisfy world is found first, and is then extended to the other worlds. Although $\mathcal{C}AltAlt$ and $POND$ utilize planning graphs similar to CGP and Frag-plan, in contrast to them, it only uses them to compute reachability estimates. The search itself is conducted in the space of belief states.

Another strand of work models conformant and contingent planning as a search in the space of belief states. This started with Genesereth and Nourbakhsh [1993], who concentrated on formulating a set of admissible pruning conditions for controlling search. There were no heuristics for choosing among unpruned nodes. GPT [Bonet and Geffner, 2000] extended this idea to consider a simple form of reachability heuristic. Specifically, in computing the estimated cost of a belief state, GPT assumes that the initial state is fully observable. The cost estimate itself is done in terms of reachability (with relaxed dynamic programming rather than planning graphs). GPT's reachability heuristic is similar to our h^{MG}_{level} heuristic because they both underestimate the cost of the farthest (max distance) state by looking at a deterministic relaxation of the problem. In comparison to GPT, $\mathcal{C}AltAlt$ and $POND$ can be seen as using heuristics that do a better job of considering the cost of the belief state across the various possible worlds.

Another family of planners that search in belief states is the MBP-family of planners—MBP [Bertoli *et al.*, 2001a], CMBP [Cimatti and Roveri, 2000], HSCP [Bertoli *et al.*, 2001b] and KACMBP [Bertoli and Cimatti, 2002]. In comparison to $\mathcal{C}AltAlt$ and $POND$, the MBP-family of planners all represent belief states in terms of binary decision diagrams. Action application is modelled as modifications to the BDDs. CMBP and HSCP support both progression and regression in the space of belief states, while KACMBP is a purely progression planner. While CMBP concentrated on efficient BDD manipulations, HSCP employs a cardinality heuristic in addition. Before computing heuristic estimates, KACMBP pro-actively reduces the uncertainty (disjunction) in the belief state by taking actions that effectively force the agent into states with reduced uncertainty. The motivation for this approach is that applying heuristics to belief states containing multiple states may not give accurate enough direction to the search. While reducing the uncertainty seems to be an effective idea, we note that (a) not all domains may contain actions that reduce belief state uncertainty and (b) the need for uncertainty reduction may be reduced when we have heuristics that effectively reason about the multiple worlds (viz., our multiple planning graph heuristics). Nevertheless, it would be very fruitful to integrate knowledge goal ideas of KACMBP and the reachability heuristics of $\mathcal{C}AltAlt$ and $POND$ to handle domains that contain both high uncertainty and costly goals.

In contrast to these domain-independent approaches that only require models of the domain physics, PKSPlan [Bacchus, 2002] is a forward-chaining *knowledge-based planner* that requires richer domain knowledge. The planner makes use of several knowledge bases that are updated by actions, opposed to a single knowledge base taking the form of a belief state. The knowledge bases separate binary and multi-valued variables and planning and execution time knowledge.

Finally, $\mathcal{C}AltAlt$ and $POND$ are also related to, and an adaptation of the work on, reachability heuristics for classical planning, including $AltAlt$ [Nguyen *et al.*, 2002], FF [Hoffmann and Nebel, 2001] and HSP-r [Bonet and Geffner, 1999]. $\mathcal{C}AltAlt$ is the conformant extension to $AltAlt$ that uses regression search (similar to HSP-r) guided by planning graph heuristics. $POND$ is similar to FF in that it uses progression search based on planning graph heuristics.

7. Conclusion

With the intent of scaling belief space planning to domains where reachability of subgoals is a non-trivial search problem, we have:

1. Discussed what the heuristic measures should be estimating.

2. Shown how to compute such heuristic measures on planning graphs.

3. Provided empirical comparisons of these measures.

4. Learned that a labelled uncertainty graph can capture the same support information as multiple graphs, and reduces the cost of heuristic computation.

5. Learned that the labelled uncertainty graph is very useful for conformant planning and, without considering observational actions and knowledge, can perform well in contingent planning.

We've shown that planning with a Labelled Uncertainty planning Graph LUG, a condensed version of the multiple graphs is useful for encoding conformant reachability information. Our main innovation is the idea of "labels" – labels are attached to all literals, actions, effect relations, and mutexes to indicate the set of worlds in which those respective elements hold. Our experimental results show that the LUG can outperform the multiple graph approach by reducing memory requirements and heuristic computation time. In comparison to other approaches, we've also been able to demonstrate the utility of structured reachability heuristics in controlling plan length for both conformant and contingent planning.

We intend to investigate two additions to this work. The first, and easier, addition is to incorporate sensing and knowledge into the heuristics. We already have some promising results without these features in the planning graphs, but hope that they will help the approaches scale even further on contingent problems. We see a regression contingent planner, like the planner described in [Rintanen, 2003], being the best approach because, as in C*AltAlt* the planning graph would be built once per problem, rather than for each search node.

The second addition will be to consider heuristics for stochastic planning problems. We foresee that planners such as Buridan [Kushmerick *et al.*, 1994] could use these types of heuristics to create a seed plan, which would then be order generalized, and improved through adding support to weakly supported conditions. The advantage would be that these heuristics would help generate a starting plan very quickly, one that is aware of some of the non-determinism and incompleteness of the problem and much more robust than a simple classical plan.

Acknowledgments We would like to thank the many anonymous reviewers of this work, as well as Minh B. Do, Romeo Sanchez, Terry Zimmermam, Satish Kumar Thittamaranahalli, and Eric Hansen for helpful discussions and feedback, and Piergiorgio Bertoli for helping with the MBP planner. This work was supported in part by NASA grants NCC2-1225 and NAG2-1461, and the NSF grant IIS-0308139, as well as the 2003 NASA RIACS SSRP.

References

Ronald P.A. Petrick Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Artificial Intelligence Planning Systems*, pages 212–221, 2002.

Piergiogio Bertoli and Alessandro Cimatti. Improving heuristics for planning as search in belief space. In *Artificial Intelligence Planning Systems*, pages 143–152, 2002.

Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In Bernhard Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 473–486, San Francisco, CA, August 4–10 2001. Morgan Kaufmann Publishers, Inc.

Piergorgio Bertoli, Alessandro Cimatti, and Marco Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, 2001.

Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proceedings of the Euoropean Conference of Planning*, pages 360–372, 1999.

Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning Systems*, pages 52–61, 2000.

Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a bdd package. In *Conference proceedings on 27th ACM/IEEE design automation conference*, pages 40–45. ACM Press, 1990.

Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In *6th European Conference on Planning*, 2001.

Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *CAV 2002*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In K. Hammond, editor, *Proceedings of the Second International Conference on AI Planning Systems*, pages 31–36, Menlo Park, California, 1994. American Association for Artificial Intelligence.

Michael R. Genesereth and Illah R. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 724–730, Menlo Park, CA, USA, July 1993. AAAI Press.

Eric A. Hansen and Shlomo Zilberstein. LAO: A heuristic-search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2):35–62, 2001.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Volume 20, special issue on the 3rd international planning competition. *Journal of Artificial Intelligence Research*, 2003.

J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. Technical Report report00088, IBM, 1, 1997.

Jana Koehler. Handling of conditional effects and negative goals in IPP. Technical Report report00128, IBM, 17, 1999.

James Kurien, P. Pandurang Nayak, and David E. Smith. Fragment-based conformant planning. In *Artificial Intelligence Planning Systems*, pages 153–162, 2002.

Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1073–1078, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

Hector Levesque. What is planning in the presence of sensing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

Drew McDermott. A critique of pure reason. *Computational Intelligence*, 3(3):151–237, 1987.

XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.

Edwin P. D. Pednault. Synthesizing plans that contain actions with context-dependent effects. Technical Memorandum, AT&T Bell Laboratories, Murray Hill, NJ, 1987.

Jussi Rintanen. Product representation of belief spaces in planning under partial observability. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-03)*, Aculpulco, Mexico, August 2003.

David E. Smith and Daniel S. Weld. Conformant graphplan. In *(AAAI-98) and (IAAI-98)*, pages 889–896, Menlo Park, July 26–30 1998. AAAI Press.

Daniel S. Weld, Corin Anderson, and David E Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press, 1998.

Appendix A. Omelette Example

We present an example of $POND$ search where there is a cycle in the solution. The omelette problem, first described in [Levesque, 1996], is simplified here to have the objective of getting one (rather than three) good eggs in one bowl. The actions are to grab a new egg, break the held egg into a bowl, clean the contents of a bowl, and observe if a bowl contains at least one bad egg. The action descriptions are as follows:

$Grab : \{\rho_e : \neg holding,$
$\quad \rho_0 : \top \implies \varepsilon_0 : (haveGood \vee haveBad) \wedge (\neg haveGood \vee \neg haveBad)\}$
$Break : \{\rho_e : holding,$
$\quad \rho_0 : \top \implies \varepsilon_0 : \neg holding \wedge \neg empty$
$\quad \rho_1 : haveGood \implies \varepsilon_1 : good,$
$\quad \rho_2 : haveBad \implies \varepsilon_2 : bad\}$
$ObserveBad : \{\rho_e : \neg empty$
$\quad o_1 : good,$
$\quad o_2 : bad\}$
$Clean : \{\rho_e : \top,$
$\quad \rho_0 : \top \implies \varepsilon_0 : empty \wedge \neg good \wedge \neg bad\}$

The initial belief state is:
$BS_{I'} = \neg holding \wedge empty \wedge \neg good \wedge \neg bad \wedge \neg haveGood \wedge \neg haveBad$

The goal state is:
$BS_{G'} = good$ The plan (Figure 25) contains a cycle because there is no deterministic length plan that ensures that the agent grabs a good egg, rather they have to grab, break, and observe until getting a good egg. The solution is found by expanding the initial belief state to get:

$BS_{11} = Progress(BS_{I'}, Grab)$
$\quad = holding \wedge empty \wedge \neg good \wedge \neg bad \wedge$
$\quad (haveGood \vee haveBad) \wedge (\neg haveGood \vee \neg haveBad),$

then

$BS_{12} = Progress(BS_{11}, Break)$
$\quad = \neg holding \wedge \neg empty \wedge (good \vee bad) \wedge (\neg good \vee \neg bad) \wedge \neg haveGood \wedge \neg haveBad,$

and then

$B_2 = \{BS_{13}, BS_{14}\} = Progress(BS_{12}, ObserveBad)$
$\quad = \{\neg holding \wedge \neg empty \wedge good \wedge \neg bad \wedge \neg haveGood \wedge \neg haveBad,$
$\quad \neg holding \wedge \neg empty \wedge bad \wedge \neg good \wedge \neg haveGood \wedge \neg haveBad\}.$

and finally,

$BS_{15} = Progress(BS_{14}, Clean)$
$\quad = \neg holding \wedge empty \wedge \neg bad \wedge \neg good \wedge \neg haveGood \wedge \neg haveBad,$

which entails $BS_{I'}$.

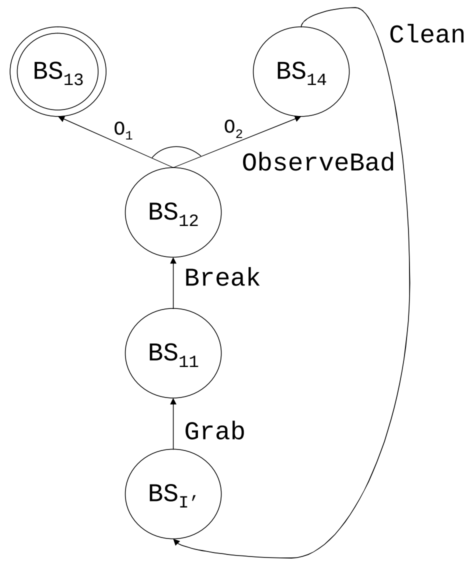

Figure 25: Illustration of progression search for a contingent plan in the omelette problem.

We add a cycle between BS_{14} and $BS_{I'}$ (without adding the redundant belief state BS_{15}).

Appendix B. Cross-World Mutexes

Mutexes can develop not only in the same possible world but also between two possible worlds, as described in [Smith and Weld, 1998]. We describe cross-world mutexes in the appendix because we implemented them but found that the cost of computing them outweighed their benefit.

Cross-world mutexes appear in our running example $CBTC$. For instance, $\hat{\mathcal{L}}_1$ can have a cross-world mutex between $\neg arm$ in one world and $\neg arm$ in the other world. This mutex arises from the fact that the two $Dunk$ actions – the only ways to support $\neg arm$ – cannot both be performed at level zero because they negate each-other's executability precondition. The mutex is cross-world because the respective conditional effects that give $\neg arm$ are supported in different possible worlds. Checking only same-world mutexes, we would not find this mutex and think $\neg arm$ is reachable after one step, as without using any mutexes.

The representation of cross-world mutexes requires another generalization for the labelling of mutexes. Same world mutexes require keeping only one label for the mutex to signify all same possible worlds for which the mutex holds. The extended representation keeps a pair of labels, one for each element in the mutex; if x in possible world S is mutex with x' in possible world S', we denote the mutex as the pair ($\hat{\ell}_k(x) = S, \hat{\ell}_k(x') = S'$).

For the example of $\neg arm$ in the previous paragraph, the mutex would be ($\hat{\ell}_1(\neg arm) = inP1 \wedge \neg inP2, \hat{\ell}_1(\neg arm) = \neg inP1 \wedge inP2$).

The use of cross-world mutexes requires an extension of the definitions for fully-mutex-supported and mutex-supported. We need to consider that for any pair of possible worlds there may exist a mutex that makes a formula f not supported.

A formula f is **fully-cross-mutex-supported** ($FSp^{CX}(f,k)$) at level k for all possible worlds of BS_P when there are no two possible worlds S and S' that entail BS_P where f is not supported for both worlds.

A formula f in possible world S and f' in possible world S' are **cross-mutex-supported** ($Sp^{CX}(f,S,f',S',k)$) at level k when (i) the labels of the literals in f indicate f is supported in both S and likewise for f' in S', and (ii) there is no cross-world mutex between the two possible worlds S and S' for f and f'. For Sp^{CX} to hold for both f and f', Sp^{NX} must hold for both f and f', and there must not exist a mutex between f and f'. To ease the definition, we consider a canonical form for f (and f'), namely a CNF, $\mathcal{C}$ (and $\mathcal{C}'$), that are supported when:

$$Sp^{NX}(\mathcal{C}, S, k) \wedge Sp^{NX}(\mathcal{C}', S', k)$$

and

$$\neg\exists_{\substack{C\in\mathcal{C}\\ C'\in\mathcal{C}'}} \forall_{\substack{l\in C\\ l'\in C'}} \exists_{(\hat{\ell}_k(l),\hat{\ell}_k(l'))\in\hat{\mathcal{L}}_k} S \models \hat{\ell}_k(l) \wedge S' \models \hat{\ell}_k(l') \tag{A-1}$$

The computation of cross-world mutexes requires changes to some of the mutex formulas, as outlined next. The major change is to check, instead of all the single possible worlds S, all pairs of possible worlds S and S' for mutexes.

Action Mutexes $\hat{\mathcal{A}}_k$: The action mutexes can now hold for actions that are executable in different possible worlds.

- **Interference** does not change for cross-world mutexes, except that there is a pair of labels where ($\hat{\ell}_k(a) = \top, \hat{\ell}_k(a') = \top$), instead of a single label.

  An example of cross-world action interference mutexes is shown in Figure 26. The discussion surrounding the reasons for mutexes is identical to the example in Figure 9, with the exception that there are two labels to each mutex, namely ($\top, \top$).

- **Competing Needs** changes for cross-world mutexes because two actions a and a', in worlds S and S' respectively, could be competing. Formally, a cross-world competing needs mutex (($\hat{\ell}_k(a) = S, \hat{\ell}_k(a') = S'$) exists between a and a' in worlds S and S' if:

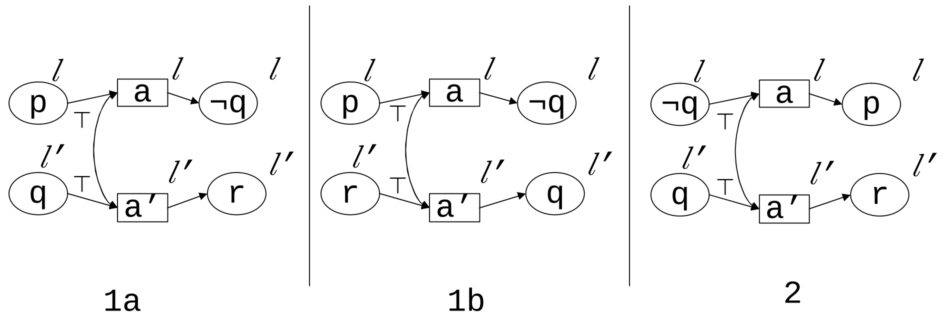

Figure 26: Example of a cross-world action interference mutex.

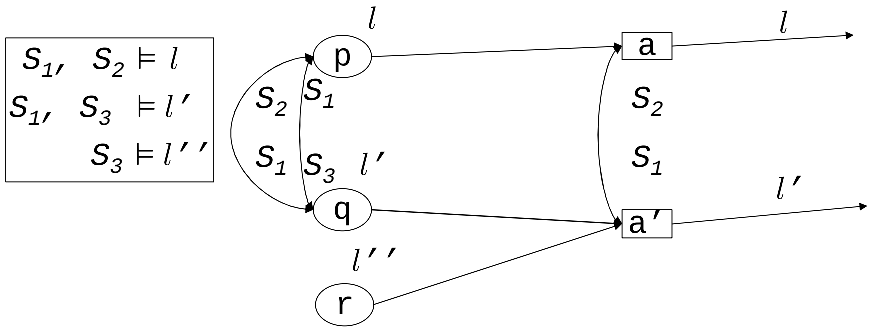

Figure 27: Example of a cross-world action competing needs mutex.

$$\neg Sp^{CX}(\rho_e, S, \rho'_e, S', k) \tag{A-2}$$

An example of a cross-world action competing needs mutex is illustrated in figure 27. Literal p holds in possible worlds S_1 and S_2 (denoted by label ℓ), q holds in possible worlds S_1 and S_3 (denoted by label ℓ'), and r holds in world S_3 (denoted by ℓ"), but p and q cannot hold together when they hold in the possible worlds (S_1, S_3) and (S_2, S_1) because they are mutex across those possible worlds. The action a' has the enabling precondition $q \vee r$. When checking for a competing needs mutex between a and a', we see that the only pair of possible worlds (S, S') where $\neg Sp^{CX}(p, S, (q \vee r), S', k)$ holds is (S_2, S_1) because even though q and p are cross-world mutex for (S_1, S_3) and (S_2, S_1) r can support a' in S_3.

Effect Mutexes $\hat{\mathcal{E}}_k$: The effect mutexes can now hold for effects that occur in different possible worlds.

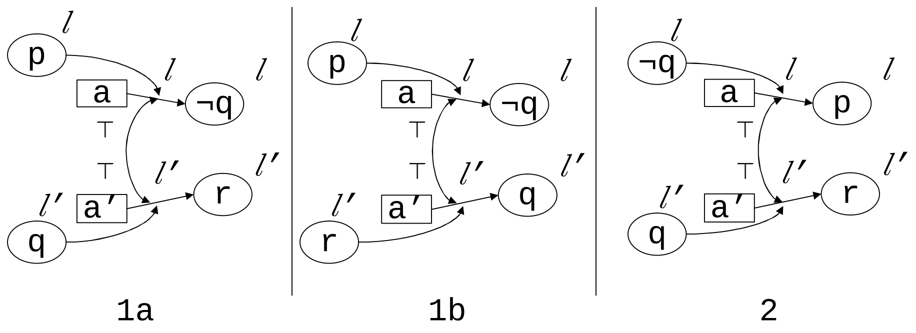

Figure 28: Example of a cross-world effect interference mutex.

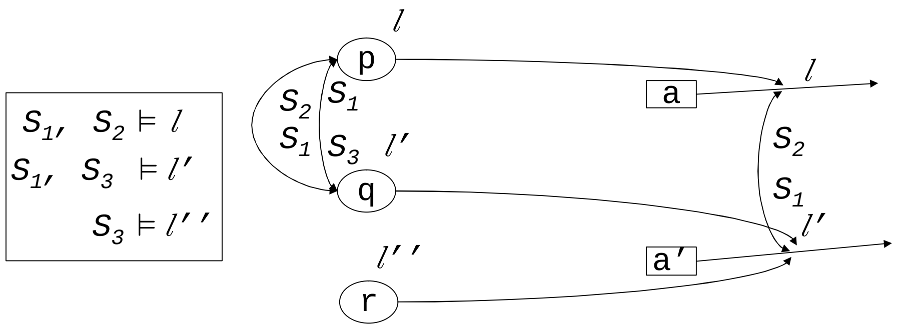

Figure 29: Example of a cross-world effect competing needs mutex.

- **Interference** does not change for cross-world mutexes, except that there is a pair of labels where $(\hat{\ell}_k(\varphi_i) = \top, \hat{\ell}_k(\varphi'_j) = \top)$, instead of a single label.

  An example of cross-world effect interference mutexes is shown in Figure 28. The discussion surrounding the reasons for mutexes is identical to the example in Figure 11, with the exception that there are two labels to each mutex, namely $(\top, \top)$.

- **Competing Needs** changes for cross-world mutexes because two effects φ_i of a and φ'_j of a', in worlds S and S' respectively, could be competing. Formally, a cross-world competing needs mutex $(\hat{\ell}_k(\varphi_i) = S, \hat{\ell}_k(\varphi'_j) = S')$ exists between φ_i of a and φ'_j of a' in worlds S and S' if:

$$\neg Sp^{CX}(\rho_i, S, \rho'_j, S', k) \tag{A-3}$$

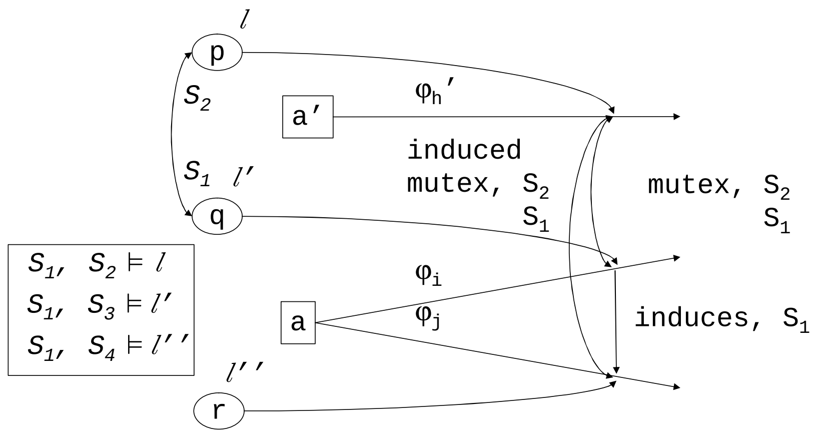

Figure 30: Example of a cross-world induced effect mutex.

An example of a cross-world effect competing needs mutex is illustrated in figure 29. Literal p holds in possible worlds S_1 and S_2 (denoted by label ℓ), q holds in possible worlds S_1 and S_3 (denoted by label ℓ'), and r holds in world S_3 (denoted by ℓ''), but p and q cannot hold together when they hold in the possible worlds (S_1, S_3) and (S_2, S_1) because they are mutex across those possible worlds. The conditional effect of action a' has the antecedent $q \vee r$. When checking for a competing needs mutex between the conditional effects of a and a', we see that the only pair of possible worlds (S, S') where $\neg Sp^{CX}(p, S, (q \vee r), S', k)$ holds is (S_2, S_1) because even though q and p are cross-world mutex for (S_1, S_3) and (S_2, S_1) r can support the antecedent of the conditional effect of a' in S_3.

- **Induced** mutexes change slightly for cross-world mutexes. The formula f, representing the worlds where one effect induces another, remains the same, but the mutex changes slightly. If there exists a mutex $(\hat{\ell}_k(\varphi_i), \hat{\ell}_k(\varphi'_h))$, and φ_i induces φ_j, then the mutex $(\hat{\ell}_k(\varphi_j) = f \wedge \hat{\ell}_k(\varphi_i), \hat{\ell}_k(\varphi'_h) = \hat{\ell}_k(\varphi'_k))$ holds.

  An example of a cross-world induced effect mutex is shown in Figure 30. Literal p holds in possible worlds S_1 and S_2 (denoted by label ℓ), q holds in possible worlds S_1 and S_3 (denoted by label ℓ'), and r holds in worlds S_1 and S_4 (denoted by ℓ''). Literals p and q are mutex across possible worlds (S_2, S_1). The effect φ_i of a induces φ_j in possible world S_1, φ'_h is mutex with effect φ_i across possible worlds (S_2, S_1)

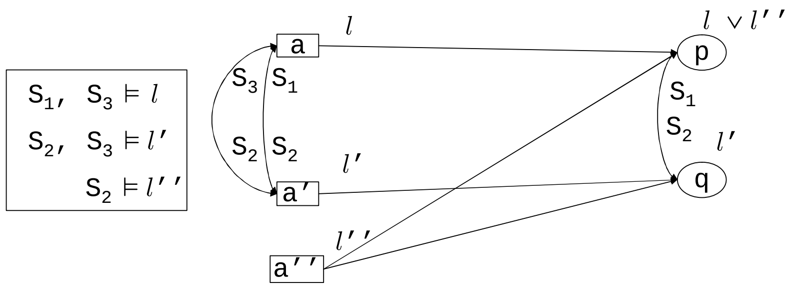

Figure 31: Example of a cross-world inconsistent support mutex.

because of the mutex between p and q, and φ_j becomes induced mutex with φ'_h across possible world (S_2, S_1).

Literal Mutexes $\hat{\mathcal{L}}_k$: The literal mutexes can now hold for literals that are supported in different possible worlds.

- **Inconsistent Support** changes for cross-world mutexes. A mutex ($\hat{\ell}_k(l) = S, \hat{\ell}_k(l') = S'$) holds for l in S and l' in S' if:

$$\neg Sp^{CX}(l, S, l', S', k) \tag{A-4}$$

An example of a cross-world inconsistent literal mutex is shown in Figure 31. Action a has an effect that is supported in worlds S_1 and S_3 (denoted by ℓ) and gives p, action a' has an effect that is supported in worlds S_2 and S_3 (denoted by ℓ') and gives q, and action a'' has an effect supported in world S_2 (denoted by ℓ'') and gives $p \wedge q$. There is a cross-world effect mutex between the effects of a and a' across worlds (S_1, S_2) and (S_3, S_2). The only possible worlds (S, S') where literals p and q are $\neg Sp^{CX}(pS, q, S', k)$ is (S_1, S_2).

Appendix C. Sample $POND$ Plan

Following is a sample plan constructed by $POND$ using the h^{LUG}_{RP-ha} heuristic. The plan is for the $Rovers$ domain, problem 6. This is one of the more difficult problems we used for evaluation, and we show it here to give the reader a better sense of the size of the problems $POND$ can solve. MBP was the only other planner that could find a solution for this problem, but its solution had a maximum branch length of 158 actions, compared to $POND$'s 23.

```
1: (navigate rover0 waypoint3 waypoint0 camera0 objective1)
2: (sense_vis rover0 objective1 waypoint0)

     IF: (NOT visible_from_objective1_waypoint0  )
     3: (navigate rover0 waypoint0 waypoint3 camera0 objective1)
     4: (navigate rover0 waypoint3 waypoint1 camera0 objective1)
     5: (navigate rover0 waypoint1 waypoint4 camera0 objective1)
     6: (sense_vis rover0 objective1 waypoint4)

          IF: (NOT visible_from_objective1_waypoint4  )
          7: (sense_rock rover0 objective1 waypoint4)

               IF: (NOT at_rock_sample_waypoint4  )
               8: (sense_soil rover0 objective1 waypoint4)

                    IF: (NOT at_soil_sample_waypoint4  )
                    9: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    10: (calibrate rover0 camera0 objective1 waypoint5)
                    11: (take_image rover0 waypoint5 objective1 camera0 high_res)
                    12: (sample_soil rover0 rover0store waypoint5)
                    13: (drop rover0 rover0store)
                    14: (sample_rock rover0 rover0store waypoint5)
                    15: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    16: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
                    17: (communicate_image_data rover0 general objective1 high_res
                         waypoint1 waypoint0)
                    18: (communicate_soil_data rover0 general waypoint1 waypoint0)
                    19: (communicate_rock_data rover0 general waypoint1 waypoint0)
                    DONE

                    IF: at_soil_sample_waypoint4
                    20: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    21: (calibrate rover0 camera0 objective1 waypoint5)
                    22: (take_image rover0 waypoint5 objective1 camera0 high_res)
                    23: (sample_rock rover0 rover0store waypoint5)
                    24: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    25: (drop rover0 rover0store)
                    26: (sample_soil rover0 rover0store waypoint4)
                    GOTO 16

               IF: at_rock_sample_waypoint4
               27: (sense_soil rover0 objective1 waypoint4)

                    IF: (NOT at_soil_sample_waypoint4  )
                    28: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    29: (calibrate rover0 camera0 objective1 waypoint5)
                    30: (take_image rover0 waypoint5 objective1 camera0 high_res)
                    31: (sample_soil rover0 rover0store waypoint5)
                    32: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    33: (drop rover0 rover0store)
                    34: (sample_rock rover0 rover0store waypoint4)
                    GOTO 16

                    IF: at_soil_sample_waypoint4
                    35: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    36: (calibrate rover0 camera0 objective1 waypoint5)
                    37: (take_image rover0 waypoint5 objective1 camera0 high_res)
                    38: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    39: (sample_soil rover0 rover0store waypoint4)
                    GOTO 33

          IF: visible_from_objective1_waypoint4
          40: (sense_rock rover0 objective1 waypoint4)

               IF: (NOT at_rock_sample_waypoint4  )
               41: (sense_soil rover0 objective1 waypoint4)

                    IF: (NOT at_soil_sample_waypoint4  )
                    42: (calibrate rover0 camera0 objective1 waypoint4)
                    43: (take_image rover0 waypoint4 objective1 camera0 high_res)
                    44: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    45: (sample_soil rover0 rover0store waypoint5)
                    46: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    47: (drop rover0 rover0store)
                    48: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
                    49: (communicate_image_data rover0 general objective1 high_res
                         waypoint1 waypoint0)
                    50: (communicate_soil_data rover0 general waypoint1 waypoint0)
                    51: (navigate rover0 waypoint1 waypoint4 camera0 objective1)
                    52: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    53: (sample_rock rover0 rover0store waypoint5)
                    54: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    55: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
                    56: (communicate_rock_data rover0 general waypoint1 waypoint0)
                    DONE

                    IF: at_soil_sample_waypoint4
                    57: (calibrate rover0 camera0 objective1 waypoint4)
                    58: (take_image rover0 waypoint4 objective1 camera0 high_res)
                    59: (sample_soil rover0 rover0store waypoint4)
                    GOTO 47

               IF: at_rock_sample_waypoint4
               60: (sense_soil rover0 objective1 waypoint4)

                    IF: (NOT at_soil_sample_waypoint4  )
                    61: (calibrate rover0 camera0 objective1 waypoint4)
                    62: (take_image rover0 waypoint4 objective1 camera0 high_res)
                    63: (sample_rock rover0 rover0store waypoint4)
                    64: (drop rover0 rover0store)
                    65: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
                    66: (communicate_image_data rover0 general objective1 high_res
                         waypoint1 waypoint0)
                    67: (communicate_rock_data rover0 general waypoint1 waypoint0)
                    68: (navigate rover0 waypoint1 waypoint4 camera0 objective1)
                    69: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
                    70: (sample_soil rover0 rover0store waypoint5)
                    71: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
                    72: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
                    73: (communicate_soil_data rover0 general waypoint1 waypoint0)
                    DONE

                    IF: at_soil_sample_waypoint4
                    74: (calibrate rover0 camera0 objective1 waypoint4)
                    75: (take_image rover0 waypoint4 objective1 camera0 high_res)
                    76: (sample_soil rover0 rover0store waypoint4)
```

```
               77: (drop rover0 rover0store)
               78: (sample_rock rover0 rover0store waypoint4)
               79: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
               80: (communicate_image_data rover0 general objective1 high_res
                    waypoint1 waypoint0)
               81: (communicate_soil_data rover0 general waypoint1 waypoint0)
               GOTO 56

IF: visible_from_objective1_waypoint0
82: (calibrate rover0 camera0 objective1 waypoint0)
83: (take_image rover0 waypoint0 objective1 camera0 high_res)
84: (navigate rover0 waypoint0 waypoint3 camera0 objective1)
85: (navigate rover0 waypoint3 waypoint1 camera0 objective1)
86: (communicate_image_data rover0 general objective1 high_res
     waypoint1 waypoint0)
87: (navigate rover0 waypoint1 waypoint4 camera0 objective1)
88: (sense_rock rover0 objective1 waypoint4)

     IF: (NOT at_rock_sample_waypoint4  )
     89: (sense_soil rover0 objective1 waypoint4)

          IF: (NOT at_soil_sample_waypoint4  )
          90: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
          91: (sample_soil rover0 rover0store waypoint5)
          92: (drop rover0 rover0store)
          93: (sample_rock rover0 rover0store waypoint5)
          94: (navigate rover0 waypoint5 waypoint4 camera0 objective1)
          95: (navigate rover0 waypoint4 waypoint1 camera0 objective1)
          96: (communicate_soil_data rover0 general waypoint1 waypoint0)
          97: (communicate_rock_data rover0 general waypoint1 waypoint0)
          DONE

          IF: at_soil_sample_waypoint4
          98: (sample_soil rover0 rover0store waypoint4)
          99: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
          GOTO 92

     IF: at_rock_sample_waypoint4
     100: (sense_soil rover0 objective1 waypoint4)

          IF: (NOT at_soil_sample_waypoint4  )
          101: (sample_rock rover0 rover0store waypoint4)
          102: (drop rover0 rover0store)
          103: (navigate rover0 waypoint4 waypoint5 camera0 objective1)
          104: (sample_soil rover0 rover0store waypoint5)
          GOTO 94

          IF: at_soil_sample_waypoint4
          105: (sample_soil rover0 rover0store waypoint4)
          106: (drop rover0 rover0store)
          107: (sample_rock rover0 rover0store waypoint4)
          GOTO 95
```