

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

Some recent works in conditional planning have proposed reachability heuristics to improve planner scalability, but many lack a formal description of the properties of their distance estimates. To place previous work in context and extend work on heuristics for conditional planning, we provide a formal basis for distance estimates between belief states. Our discussion surrounds a definition for the distance between belief states that relies on aggregating underlying state distance measures. We analyze several techniques to aggregate state distances and their associated properties. Many existing heuristics exhibit a subset of the properties, but in order to provide a standardized comparison we present several generalizations of planning graph heuristics that are used in a single planner. We compliment our belief state distance estimate framework by also investigating efficient planning graph data structures that incorporate BDDs to compute the most effective heuristics.

We developed two planners to serve as test-beds for our investigation. The first, *CAltAlt*, is a conformant regression planner that uses A* search. The second, *POND*, is a conditional progression planner that uses AO* search. 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 conditional 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], PKSPan [Bacchus, 2002], Frag-Plan [Kurien *et al.*, 2002], MBP [Bertoli *et al.*, 2001a], KACMBP [Bertoli and Cimatti, 2002], CFF [Brafman and Hoffmann, 2004], and YKA [Rintanen, 2003]. Several

of these planners are extensions of heuristic state search planners that search in the space of uniform probability “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 issue of what the heuristics estimate 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 a destination belief state. The question of course is how to compute these measures and which are most effective. We argue that many existing works do not adequately address this question. Many approaches estimate belief state distances in terms of individual state to state distances between states in two belief states, but either lack effective state to state distances or ways to aggregate the state distances. For instance the MBP planner [Bertoli *et al.*, 2001a] measures the number of states in the current belief, assuming each state distance has unit cost, is independent, and using a summation. The GPT planner [Bonet and Geffner, 2000] measures the state to state distances with a reachability heuristic and takes the max distance, assuming the states of the belief positively interact.

We evaluate many of these different approaches to estimating belief state distance in terms of state to state distances. The basis of our investigation is in adapting classical planning reachability heuristics to measure state distances and developing state distance aggregation techniques to measure interaction between plans for states in a belief. We take three fundamental approaches to measure the distance between two belief states. The first approach does not involve aggregating state distance measures, rather we approximate the beliefs with two states and use a classical planning graph to make a state distance measure between the states. The second retains the distinction between individual states in the beliefs and using multiple planning graphs (akin to CGP [Smith and Weld, 1998]) computes many state distance measures which are then aggregated. The third employs a new planning graph generalization, called the Labelled Uncertainty Graph (*LUG*), that blends the first two to measure a single distance between two belief states. With each of these techniques we will discuss the types of heuristics that we can compute but focus on relaxed plans. We present several relaxed plans that differ in terms of how they employ state distance aggregation to make stronger assumptions about state independence, positive interaction, and negative interaction.

Our motivation for the three techniques for measuring belief state distances is to, as in the first, try a minimal extension to classical planning heuristics to see if they will work for us. Noticing that our use of classical planning heuristics ignores distinctions between states in a belief and may provide uninformed heuristics, we move to the second approach where we possibly build exponentially many more planning graphs to get a better heuristic. With the multiple planning graphs we will extract a heuristic from each graph and aggregate them

to get the belief state distance measure. If we assume the states of a belief are independent, we can aggregate the measures with a summation. Or, if we assume they positively interact we can use a maximization. However, as we will show, relaxed plans give us a unique opportunity to measure both positive interaction and independence among the states by essentially taking the union of relaxed plans. Moreover, mutexes play a role in measuring negative interactions between states. Despite the utility in having robust ways to aggregate state distances, we are still faced with the exponential blow up in the number of planning graphs needed. Thus, our third approach seeks to retain the ability to measure interaction of state distances but avoid computing multiple graphs and extracting heuristics from each. The idea is to condense the symbolically represent multiple planning graphs in 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 (i.e. initial literal layers), into “labels”. The planning graph vertices are the same as those present in multiple graphs, but redundant representation is avoided. 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 planning graph projection from each possible world. We will describe how to extract heuristics from the *LUG* that make implicit assumptions about state interaction without explicitly aggregating several state distances.

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 new heuristics developed around the notion of overlap (multiple world positive interaction and independence). We implemented the heuristics within two planners, the Conformant-*AltAlt* planner (*CAltAlt*) and the Partially-Observable Non-Deterministic planner (*POND*).¹

We focus our attention on finding strong plans (i.e. plans that succeed with probability 1) given an uncertain initial state (with uniform probability over possible states). Sensing actions give partial observations, causative actions have deterministic conditional effects, all actions have associated costs, and the model uses a factored representation.

1. *POND* is an acronym that includes “non-deterministic” and the planner does handle search with non-deterministic actions, but we refrain from evaluating problems with non-deterministic actions in this work.

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 CFF, and our planners are competitive with BDD-based planners such as MBP and YKA, and GraphPlan-based ones such as CGP and SGP. We also notice that our planners gain scalability with our heuristics and do not give up reasonable solution quality, a property unlike several of the planners we compare to.

The rest of this paper is organized as follows. We present our work by first explaining the state and action representation used within *CAIAlt* 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 Space Planners

Our planning formulation uses regression search to find strong conformant plans and progression search to find strong conformant and conditional plans. A strong plan guarantees that after a finite number of actions executed from any of the many possible initial states, all resulting states will satisfy the goals. Conformant plans are a special case where the plan is equivalent to a simple sequence, as in classical planning. Conditional plans are a more general case where the plan is structured as a graph because they include sensory actions. In this presentation, we restrict conditional plans to DAGs, but there is no conceptual reason why they cannot be general graphs like policies. Our plan quality metric is the maximum plan path length – both causative actions and sensory action contribute to the length of a path, and sensory actions contribute to multiple paths.

We formulate search in the space of belief states, a technique described by Bonet and Geffner [2000]. The planning problem P is defined as the tuple $\langle D, BS_I, BS_G \rangle$, where D is a domain description, BS_I is the initial belief state, and BS_G is the goal belief state. The domain D is a tuple $\langle F, A \rangle$, where F is a set of all fluents and A is a set of actions.

Logical Formula Representation: We make use of logical formulas over F extensively in our approach to represent belief states, actions, and *LUG* labels, so we first explain a few conventions. We refer to the set of models of a formula f as $\mathcal{M}(f)$. We consider the disjunctive normal form of a logical formula f , $\hat{\xi}(f)$, and the conjunctive normal form of f , $\kappa(f)$. The DNF is seen as a disjunction of “constituents” \hat{S} each of which is a conjunction of literals. Alternatively the CNF is seen as a conjunction of “clauses” C each of which is

a disjunction of literals.² We find it useful to think of DNF and CNF represented as sets – a disjunctive set of constituents or a conjunctive set of clauses. We also refer the complete representation $\xi(f)$ of a formula f as a DNF where every constituent – or in this case state S – is a model of f .

Belief State Representation: A world state, S , is represented as a complete interpretation over fluents. We also refer to states as possible worlds. A belief state BS is a set of states and is symbolically represented as a propositional formula over F . A state S is in the set of states represented by a belief state BS if $S \in \mathcal{M}(BS)$.

We use the bomb and toilet with clogging and sensing problem, BTCS, as a running example for this paper.³ BTCS is a problem that includes two packages, one of which contains a bomb, and there is also a toilet in which we can dunk packages to defuse potential bombs. The goal is to disarm the bomb and the only allowable actions are dunking a package in the toilet (DunkP1, DunkP2), flushing the toilet after it becomes clogged from dunking (Flush), and using a metal-detector to sense if a package contains the bomb (DetectMetal). The fluents encoding the problem denote that the bomb is armed (arm) or not, the bomb is in a package (inP1, inP2) or not, and that the toilet is clogged (clog) or not. We also consider a conformant variation on BTCS, called BTC, where there is no sensory action.

The belief state representation of BTCS’s initial condition, in clausal representation is: $\text{arm} \wedge \neg \text{clog} \wedge (\text{inP1} \vee \text{inP2}) \wedge (\neg \text{inP1} \vee \neg \text{inP2})$, and constituent representation is: $\hat{\xi}(BS_I) = (\text{arm} \wedge \neg \text{clog} \wedge \text{inP1} \wedge \neg \text{inP2}) \vee (\text{arm} \wedge \neg \text{clog} \wedge \neg \text{inP1} \wedge \text{inP2})$. The goal in BTCS has the clausal representation: $\kappa(BS_G) = \neg \text{arm}$, and constituent representation: $\hat{\xi}(BS_G) = \neg \text{arm}$. However, the goal has the complete representation: $\xi(BS_G) = (\neg \text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg \text{inP2}) \vee (\neg \text{arm} \wedge \text{clog} \wedge \neg \text{inP1} \wedge \text{inP2}) \vee (\neg \text{arm} \wedge \neg \text{clog} \wedge \text{inP1} \wedge \neg \text{inP2}) \vee (\neg \text{arm} \wedge \neg \text{clog} \wedge \neg \text{inP1} \wedge \text{inP2}) \vee (\neg \text{arm} \wedge \text{clog} \wedge \neg \text{inP1} \wedge \neg \text{inP2}) \vee (\neg \text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \text{inP2}) \vee (\neg \text{arm} \wedge \neg \text{clog} \wedge \neg \text{inP1} \wedge \neg \text{inP2}) \vee (\neg \text{arm} \wedge \neg \text{clog} \wedge \text{inP1} \wedge \text{inP2})$.

Action Representation: We represent actions as having strictly causative or observational effects, the former termed as causative actions, the latter termed as sensory actions. All actions a are described in terms of an execution precondition $\rho^e(a)$, and a set of effects, $\Phi(a)$. The execution precondition, $\rho^e(a)$, is a conjunction of literals that must hold for the action to be executable.

Causative actions have a set of conditional effects where each conditional effect $\varphi^j(a)$ is of the form $\rho^j(a) \implies \varepsilon^j(a)$, where the antecedent ρ_j and consequent $\varphi^j(a)$ are both

-
2. It is easy to see that $\mathcal{M}(f)$ and $\hat{\xi}(f)$ are readily related. Specifically each constituent contains k of the $|F|$ literals, corresponding to $2^{|F|-k}$ models.
 3. We are aware of the negative publicity associated with the B&T problems and we do in fact handle more interesting problems with difficult reachability and uncertainty (e.g. Logistics and Rovers), but to simplify our discussion we choose this small problem.

a conjunction of literals. We handle disjunction in $\rho^e(a)$ or a $\rho^j(a)$ by replicating the respective action or effect with different conditions, so with out loss of generality we assume conjunctive preconditions. However, we cannot split disjunction in the effects because the effect would be non-deterministic. Hence we do not allow disjunction in effects thereby restricting to deterministic effects. By convention the unconditional effects are described as $\rho^0(a) = \top$ and a given $\varepsilon^0(a)$.

Sensory actions have a set $\Phi(a) = \{o^0(a), \dots, o^m(a)\}$ of observational effect formulas. Each observational effect formula, $o^j(a)$, defines the properties of an outcome of the sensor. We do not observe each $o^j(a)$ to be true or false, rather we observe each $o^j(a)$ to be true (e.g. observing the truth of a fluent requires two observational effect formulas).

The causative and sensory actions for the example BTCS problem are:

DunkP1: $\langle \rho^e = \neg \text{clog}, \Phi = \{\varphi^0 = \top \implies \text{clog}, \varphi^0 = \text{inP1} \implies \neg \text{arm}\} \rangle$

DunkP2: $\langle \rho^e = \neg \text{clog}, \Phi = \{\varphi^0 = \top \implies \text{clog}, \varphi^0 = \text{inP2} \implies \neg \text{arm}\} \rangle$

Flush: $\langle \rho^e = \top, \Phi = \{\varphi^0 = \top \implies \neg \text{clog}\} \rangle$

DetectMetal: $\langle \rho^e = \top, \Phi = \{o^0 = \text{inP1}, o^1 = \neg \text{inP1}\} \rangle$

2.1 Regression

We perform regression in the *CAItAlt* 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 not inconsistent with the belief state and (ii) at least one effect consequent contains a literal that is present in a constituent of the belief state.

Following Pednault [1987], regressing a belief state BS_i over an action a , with conditional effects, involves finding the execution, 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:⁴

$$BS_{i'} = \text{Regress}(BS_i, a) = \Pi(a) \wedge \left(\bigwedge_{C \in \kappa(BS_i)} \bigvee_{l \in C} (\Sigma(a, l) \wedge IP(a, l)) \right)$$

where the

Execution formula ($\Pi(a)$) is the execution precondition $\rho^e(a)$. This is what must hold in $BS_{i'}$ for a to have been applicable.

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

Causation formula ($\Sigma(a, l)$) for a literal l w.r.t all effects $\varphi^j(a)$ 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 . The intuitive meaning is that l already held in $BS_{i'}$, or the antecedent $\rho^j(a)$ must have held in $BS_{i'}$ to make l hold in BS_i . Formally $\Sigma(a, l)$ is defined as:

$$\Sigma(a, l) = l \vee \bigvee_{i:l \in \varepsilon^i(a)} \rho^i(a)$$

Preservation formula ($IP(a, l)$) of a literal l w.r.t. all effects $\varphi^j(a)$ of action a is defined as the formula that must be true before a such that l is not violated by any effect $\varepsilon^j(a)$. The intuitive meaning is that the antecedent of every effect that is inconsistent with l could not have held in $BS_{i'}$. We use \top so that the preservation formula is not empty should no effects conflict with l . Formally $IP(a, l)$ is defined as:

$$IP(a, l) = \top \wedge \bigwedge_{i:\neg l \in \varepsilon^i(a)} \neg \rho^i(a)$$

2.2 CAltAlt

The CAltAlt planner uses the regression operator to generate children in A* search. 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: $BS_2 = \text{Regress}(BS_G, \text{DunkP1}) = \neg \text{clog} \wedge (\neg \text{arm} \vee \text{inP1})$. The first clause is the execution formula and the second clause is the causation formula for the conditional effect of DunkP1 and $\neg \text{arm}$.

Regressing BS_2 with Flush gives: $BS_4 = \text{Regress}(BS_2, \text{Flush}) = (\neg \text{arm} \vee \text{inP1})$. For BS_4 , the execution precondition of Flush is \top , the causation formula is $\top \vee \neg \text{clog} = \top$ and $(\neg \text{arm} \vee \text{inP1})$ comes through persistence in the causation formula.

Finally, $BS_9 = \text{Regress}(BS_4, \text{DunkP2}) = \neg \text{clog} \wedge (\neg \text{arm} \vee \text{inP1} \vee \text{inP2})$.

From our example, we terminate at BS_9 because $BS_I \models BS_9$. The plan is DunkP2, Flush, DunkP1.

2.3 Progression

In progression we can handle both causative and sensory actions, so in general, progressing an action a over a belief state BS generates a set of successor belief states B . There are three distinct cases in progression, the set of resulting belief states is 1) empty when the action is not applicable to BS ($BS \not\models \rho^e(a)$), 2) a single belief state if a is a causative action, or 3) possibly several belief states if a is a sensory action.

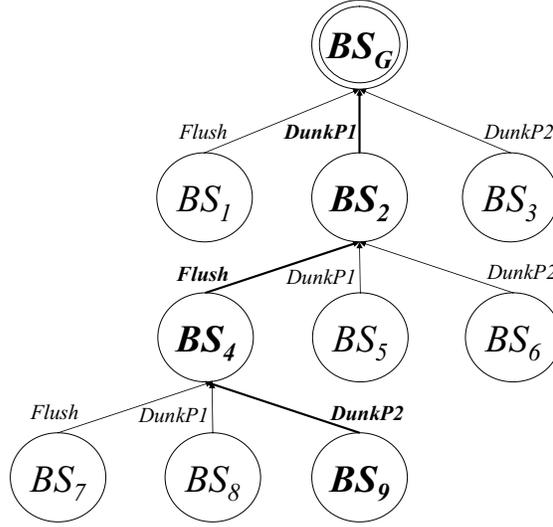


Figure 1: Illustration of the regression search path for a conformant plan in the *BTC* problem.

Progression of a belief state BS over a causative action a is best understood as the union of the result of applying a to each model of BS but we in fact implement it as BDD images, as in MBP [Bertoli *et al.*, 2001a]. Formally, progression of a causative action a over a belief state BS_i results in a set of states B , containing a single state $BS_{i'}$. If the action is applicable, the resulting belief is the disjunction of progression of each state in BS_i over a :

$$B = \{BS_{i'}\} = \text{Progress}_c(BS_i, a) = \begin{cases} \perp & : BS_i \not\models \rho^e(a) \\ \bigvee_{S \in \mathcal{M}(BS_i)} \text{Progress}_c(S, a) & : \text{otherwise} \end{cases}$$

The progression of an action a over a state S is the conjunction of every literal that persists (no applicable effect consequent contains the literal) and every literal that is given as an effect (an applicable effect consequent contains the literal).

$$S' = \text{Progress}_c(S, a) = \bigwedge_{\substack{l: l \in S \text{ and} \\ \neg \exists_j S \models \rho^j(a) \text{ and} \\ l \in \varepsilon^j(a)}} l \wedge \bigwedge_{\substack{l: \exists_j S \models \rho^j(a) \text{ and} \\ l \in \varepsilon^j(a)}} l$$

The case when an action is sensory is efficient for progression because we obtain a set of successors B by individually taking the conjunction of each observational effect $\rho^j(a)$

with BS . The progression of a sensory action a over a belief state BS_i results in a set B of belief states, defined as:

$$B = \text{Progress}_s(BS_i, a) = \begin{cases} \perp & : BS_i \not\models \rho^e(a) \\ \{BS' \mid BS' = o^j(a) \wedge BS_i\} & : \text{otherwise} \end{cases}$$

While we separate causative and sensory actions, it is possible to extend our action representation to have both types causative and observational effects. In which case, the progression would apply causative effects followed by observational effects to return a set $B = \text{Progress}_s(\text{Progress}_c(BS_i, a), a)$.

2.4 POND

We use top down AO* search [Nilsson, 1980], in the *POND* planner to generate conformant and conditional plans. In the search graph, the nodes are belief states and the hyper-edges are actions. We need AO* because the application of a sensing action to a belief state in essence partitions the belief state. We use hyper-edges for actions because sensory actions have several outcomes, all if any of which must be included in a solution.

The AO* search consists of two repeated steps, expand the current partial solution, and then revise the current partial solution. Search ends when every leaf node of the current solution is a belief state that satisfies the goal belief and no better solution exists (given our heuristic function). Expansion involves following the current solution to an unexpanded leaf node and generating its children. Revision is essentially a dynamic programming update at each node in the current solution that selects a best hyper-edge (action). The update assigns the action with minimum cost to start the best solution rooted at the given node. The cost of a node is the cost of its best action plus the average cost of its children (the nodes connected through the hyper-edge). When expanding a leaf node, the children of all applied actions are given a heuristic value to indicate their estimated cost.

The main differences between our formulation of AO* and that of Nilsson [1980] are that we do not allow cycles in the search graph, we update the costs of nodes with an average rather than a sum, and use a weighted h-value. The first difference is to ensure that plans are strong (there are a finite number of steps to the goal), the second is to guide search toward plans with lower average path cost, rather than a best worst-case execution length, and the third is to bias our search to trust the heuristic function. The reasons we define our plan quality metric differently than the metric our search tries to minimize are first that it is easier to compare to other competing planners because they measure the same plan quality metric, and second that search tends to be more efficient using the average instead of the max cost of an action's children.

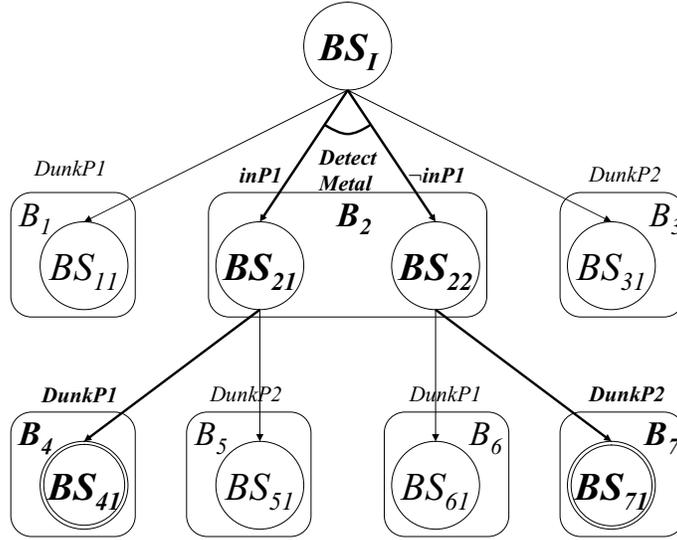


Figure 2: Illustration of progression search for a conditional plan in the BTCS problem.

As an example of search in *POND*, consider the BTCS example (Figure 2). Applying actions to the initial belief state, we get: $B_1 = \{BS_{11}\} = \text{Progress}(BS_I, \text{DunkP1}) = \{(inP1 \wedge \neg inP2 \wedge clog \wedge \neg arm) \vee (\neg inP1 \wedge inP2 \wedge clog \wedge arm)\}$, $B_2 = \{BS_{21}, BS_{22}\} = \text{Progress}(BS_I, \text{DetectMetal}) = \{inP1 \wedge \neg inP2 \wedge \neg clog \wedge arm, \neg inP1 \wedge inP2 \wedge \neg clog \wedge arm\}$, and $B_3 = \{BS_{31}\} = \text{Progress}(BS_I, \text{DunkP2}) = \{(inP1 \wedge \neg inP2 \wedge clog \wedge arm) \vee (\neg inP1 \wedge inP2 \wedge clog \wedge \neg arm)\}$.

Say we expand BS_{21} and get $B_4 = \{BS_{41}\} = \text{Progress}(BS_{21}, \text{DunkP1}) = \{inP1 \wedge \neg inP2 \wedge clog \wedge \neg arm\}$, which is a goal state, and expand BS_{22} and get $B_6 = \{BS_{61}\} = \text{Progress}(BS_{22}, \text{DunkP2}) = \{inP1 \wedge \neg inP2 \wedge clog \wedge \neg arm\}$, which is also goal state. We terminate here because this is the lowest cost plan we can find and all leaf nodes satisfy the goal. Our plan is shown in bold in Figure 2.

3. Belief State Distance

In both the *CAIAlt* and *POND* planners we need to guide search node expansion with heuristics that estimate the plan distance between a source and destination belief state. The source belief is always a search node, but the destination depends on our search direction (i.e. the initial belief in regression or the goal in progression). In either case, the distance estimate must capture how many actions are needed to transition every state in the earlier belief state to one state in the later belief state (e.g. each initial state must reach one goal state). As such, when estimating state to state distances, we optimistically assume that each

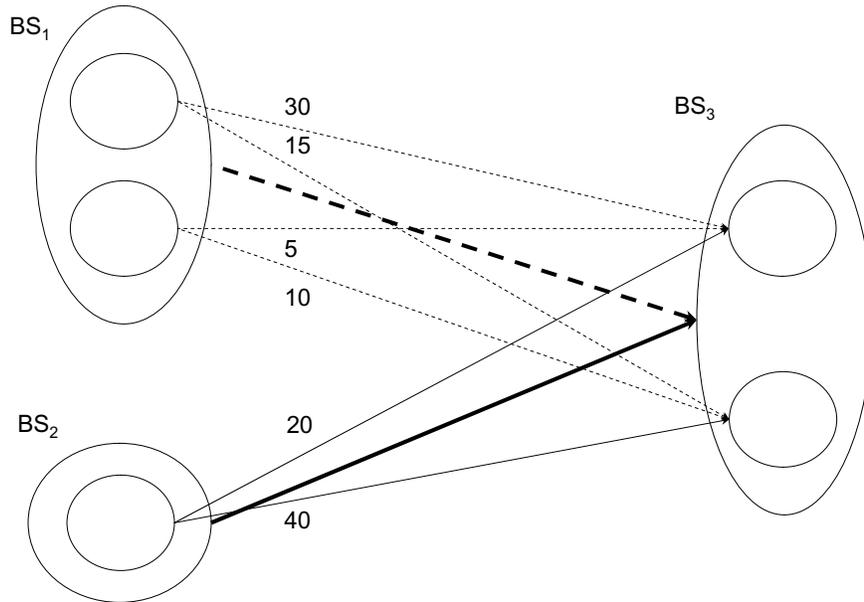


Figure 3: Conformant Plan Distance Estimation in Belief Space

of the earlier states can reach the closest of the later states (e.g. each initial state can reach its closest goal state).

Our first idea for measuring belief state distance does not measure individual state to state distances to find an aggregate distance measure, rather it aggregates the states in the two beliefs to find two approximate states. We form an approximate state by taking the union of the literals needed to express the prime implicates of the belief state formula. While this approximate state is quite possibly inconsistent, many classical planning distance measures are still applicable, and we assume these are understood. We now turn to an issue that is less understood – how to aggregate state distance measures.

Consider the example in Figure 3; there are three belief states BS_1 (containing two states), BS_2 (containing one state), and BS_3 (containing two states). The goal belief is BS_3 , and the two progression search nodes are BS_1 and BS_2 . We want to expand the search node with the smallest distance to the goal belief. The distances to estimate are between the source and destination belief states BS_1 and BS_3 (denoted by the dark, dashed line) and BS_2 and BS_3 (denoted by the dark, solid line). We’ll assume for now that we have estimates of state distance measures (denoted by the light dashed and solid lines with numbers).

There is a range of options for taking the state distances and aggregating them into a belief state distance:

- **Cardinality of Belief States:** Ignore state to state distances and count the number of states in the source belief state. This measure is akin to assuming that state to state distances are of unit distance and that making the transitions between the source and destination states is done independently without interaction. In our example the distance between BS_1 and BS_3 is two (because the two states in BS_1 need to make a transition), and the distance between BS_2 and BS_3 is one (because the one state in BS_2 needs to make a transition). The MBP family of planners [Bertoli *et al.*, 2001a] use variations on the cardinality of belief states as belief state distance estimates.
- **Positive Interaction of States:** Use state to state distances, but assume that making the transitions between source states and destination states positively interact. In our example there are two states in BS_1 , each with different distances to the states of BS_3 . We make an optimistic assumption that we can transition each source state in BS_1 to the closest destination state in BS_3 , allowing us to say the first state contributes 15 and the second state contributes 5 to the distance between BS_1 and BS_3 . The final distance estimate for BS_1 to BS_3 is 15 because we take a maximum (as is common when assuming positive interaction) over the state to state distance estimates. Likewise, for BS_2 we get 20 as the distance estimate because the state in BS_2 will optimistically reach the first state in BS_3 . The GPT planner [Bonet and Geffner, 2000] makes the positive interaction assumption in its belief state distance estimates.
- **Independence of States:** Use state to state distances, but assume the transitions between source states to destination states are independent. Again, like positive interaction, we make the optimistic assumption about transitioning to the closest destination state, but when aggregating the distance estimates for the source states we take a summation rather than maximization. In our example, we would estimate the distance from BS_1 to BS_3 as $5 + 15 = 20$, and from BS_2 to BS_3 as 20 also.
- **Overlap of States:** Use state to state distances, but account for positive interaction and independence. Assuming state to state distances are measured in terms of actual action sequences, we can analyze the actions that are common and distinct among the state to state transitions (i.e. measure overlap). For instance, if one sequence contains action a_1 , we count a_1 once, but if many sequences contain action a_2 at relatively the same time, then we can count a_2 only once. In our example, we could find that the distance from BS_1 to BS_3 is somewhere between 15 and 20, and for the distance from BS_2 to BS_3 as 20 (because there is only one state to state transition).

- **Negative Interaction of States:** Use state to state distances, but account for actions that may conflict in making different state to state transitions. Here one could make use of “cross-world” mutexes [Smith and Weld, 1998] to determine that executing one action in a specific state to state transition is mutex with an action used in another state to state transition. In our example, we may find that the distance between BS_1 to BS_3 is actually infinity if it is impossible to co-transition the states in BS_1 to a state in BS_3 .

The above techniques for belief state distance estimation in terms of state distances provide the basis for our use of multiple planning graphs. We will show in the empirical evaluation that these measures affect planner performance very differently across standard conformant and conditional planning domains. While it can be quite costly to compute several state distance measures, understanding them within the context of multiple planning graphs allows us to move toward a much cheaper implicit representation in our third idea, the *LUG*. We proceed to describe extracting belief state distance measures from planning graphs next.

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 by tracking only a subset of the infeasible tuples of literals. GraphPlan searches the planning graph once the compressed progression (or planning graph) achieves the goal literals in at a step with no two goal literals marked infeasible. The search tries to find actions to support the top level goal literals, then find actions to support those action preconditions and so on until reaching the first step. 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, CGP style planning graphs, and a new generalization of planning graphs, called the *LUG*.

Planners such as CGP [Smith and Weld, 1998] and SGP [Weld *et al.*, 1998] adapt the GraphPlan idea of compressing the search space with a planning graph by using multiple

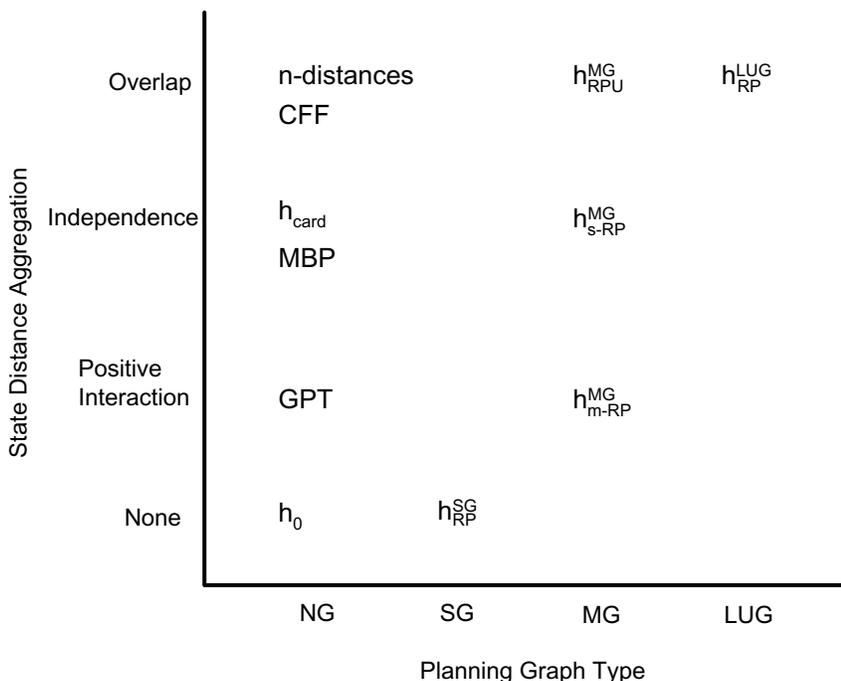


Figure 4: Taxonomy of heuristics with respect planning graph type and states cost aggregation.

planning graphs, one for each possible world in the initial belief. CGP and SGP search on these planning graphs, similar to GraphPlan, to find conformant and conditional 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.

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* heuristics are techniques based on a single classical planning graph, *MG* heuristics are techniques based on multiple planning graphs (similar to those used in CGP) and *LUG* heuristics use a new labelled planning graph. The *LUG* 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 conditional planning formulation directly uses the planning graph heuristics by ignoring observations, and our results show that this still gives good performance.

In Figure 4 we present a taxonomy of distance measures for belief space. The taxonomy also includes related planners, whose distance measures will be characterized in this

section. The figure shows how different substrates (horizontal axis) can be used to compute belief state distance by aggregating state to state distances under various assumptions (vertical axis). While there are a wealth of different heuristics one can compute using planning graphs, we concentrate on relaxed plans because they have proven the most effective in classical planning and our previous studies [Bryce and Kambhampati, 2004]. We provide additional descriptions of other heuristics like max, sum, and level in Appendix A.

To illustrate the computation of each heuristic, we use an example derived from BTC called Courteous BTC (CBTC) where a courteous package dunker has to disarm the bomb and leave the toilet unclogged, but some discourteous person has left the toilet clogged. The initial belief state of CBTC in clausal representation is $\text{arm} \wedge \text{clog} \wedge (\text{inP1} \vee \text{inP2}) \wedge (\neg\text{inP1} \vee \neg\text{inP2})$, and the goal is $\neg\text{clog} \wedge \neg\text{arm}$. The optimal action sequences to reach BS_G from BS_I are: Flush, DunkP1, Flush, DunkP2, Flush, and Flush, DunkP2, Flush, DunkP1, Flush, thus the optimal heuristic estimate for the distance between BS_I and BS_G , in regression, is $h^*(BS_G) = 5$ because in either plan there are five actions.

We use planning graphs for both progression and regression search. 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 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. We introduce a notation BS_i to denote the belief state for which we find a heuristic measure, and BS_P to denote the belief state that is used to construct the initial layer of the planning graph(s). In the following subsections we describe computing heuristics for regression, but they are generalized for progression by changing BS_i and BS_P appropriately.

We proceed by describing the various substrates used for computing belief space distance estimates. Within each we describe the prospects for various types of world aggregation. In addition to our heuristics, we mention related work in the relevant areas.

4.1 Non Planning Graph-based Heuristics (*NG*)

We group many heuristics and planners into the *NG* group because they are not using *SG*, *MG*, or *LUG* planning graphs. Sufficed to say, just because we mention them in this group, it does not mean they are not using planning graphs in some other form.

No Aggregation: Breadth first search uses a simple heuristic, h_0 where the heuristic value is set to zero. We mention this heuristic so that we can gauge the effectiveness of our search substrates relative to improvements gained through using heuristics.

Positive Interaction Aggregation: The GPT planner [Bonet and Geffner, 2000] measures belief state distance as the max state to state distance of states in the source and destination

belief states assuming optimistic reachability as mentioned in Section 3. GPT measures state distances exactly, in terms of the minimum number of transitions in the state space. Taking the max state to state distance is akin to assuming positive interaction of states in the current belief.

Independence Aggregation: The MBP planner [Bertoli *et al.*, 2001a] and our comparable h_{card} heuristic measure belief state distance by assuming every state to state distance is one, and taking the sum of the state distances (i.e. counting the number of states in a belief). This measure can be useful in regression because goal belief states are partially specified and contain many states consistent with a goal formula and many of the states consistent with the goal formula are not reachable from the initial belief. Throughout regression, many of the unreachable states are removed from successor belief states because they are inconsistent with the preconditions of a regressed action. Thus, belief states can reduce in size during regression and their cardinality may indicate they are closer to the initial belief. Cardinality is also useful in progression because as belief states become smaller, the agent has more knowledge and it can be easier to reach a goal state.

In CBTC, $h_{card}(BS_G) = 4$ because BS_G has four 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 $inP1 \wedge inP2 \wedge \neg clog \wedge \neg arm$. Counting unreachable states may overestimate the distance estimate because we do not need to plan for them. In general, aside from counting unreachable states, cardinality does not accurately reflect distance measures – consider the extreme case of having complete state information and assigning each state a distance of one.

Overlap Aggregation: Rintanen [2004] describes n-Distances which generalize the belief state distance measure in GPT to consider the max n-tuple state distance. It measures for each n-sized tuple of states in a belief the length of the actual plan to transition the n-tuple to the destination belief. Then the max n-tuple distance is taken as the distance measure.

The CFF planner [Brafman and Hoffmann, 2004] searches forward in the space of implicit belief states by representing the known (entailed) facts of a belief state together with the action history. CFF projects a planning graph from their implicit belief state to reach the goal and backwards toward the initial belief. The reason CFF must do this is because they use an implicit belief and need information from the initial belief state to capture all uncertainty. A relaxation that CFF uses is to ignore all but one antecedent literal in conditional effects to keep their relaxed plan reasoning tractable. The CFF relaxed plan represents supporting the goal from every state in the initial belief state and does capture overlap.

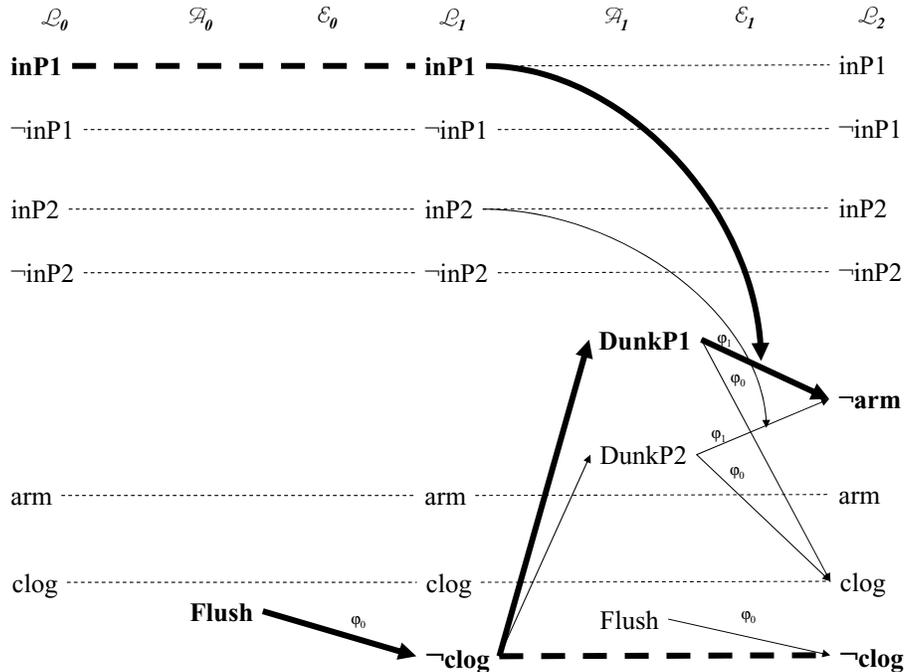


Figure 5: Single planning graph for CBTC, with relaxed plan components in bold. Mutexes omitted.

4.2 Single Graph Heuristics (SG)

The simplest approach for using planning graphs for belief space planning heuristics is to use a “classical” planning graph. We take all the literals in all models of the projected belief state and insert each literal into the initial layer of the planning graph, ignoring possible worlds. For example, in CBTC assuming regression search with $BS_P = BS_I$, the initial level of the planning graph is $\{\text{arm}, \text{clog}, \text{inP1}, \text{inP2}, \neg\text{inP1}, \neg\text{inP2}\}$ (see Figure 5).

Graph construction is identical to classical planning graphs (including mutex propagation) and stops when two subsequent literal layers are identical (level off). We use the planning graph formalism used in IPP [Koehler, 1999] to allow for explicit representation of conditional effects, meaning there is a literal layer \mathcal{L}_k , an action layer \mathcal{A}_k , and an effect layer \mathcal{E}_k in each level k . While the planning graph contains effect layers, for clarity we do not represent it explicitly in the figure. A literal is in \mathcal{L}_k if it persists from the previous literal layer \mathcal{L}_{k-1} , or an effect from the previous effect layer \mathcal{E}_{k-1} contains the literal in its consequent. An action is in the action layer \mathcal{A}_k if every one of its execution precondition literals is in \mathcal{L}_k . An effect is in the effect layer \mathcal{E}_k if its associated action is in the action layer \mathcal{A}_k and every one of its antecedent literals is in \mathcal{L}_k . Using conditional effects in the

planning graph avoids factoring an action with conditional effects into a possibly exponential number of non-conditional actions, but adds an extra planning graph layer per level. Once our graph is built, we can extract heuristics.

No Aggregation: Relaxed plans within a single planning graph are able to measure, under the most optimistic assumptions, the distance between two beliefs. Since the single graph has an initial literal layer of all literals in a belief, the relaxed plan represents a distance between a state (comprised of a subset of the initial layer literals) and a state (comprised of the literals in a constituent of our belief state). In relaxed plans we cannot ensure that any given subset of the initial layer literals is used to support the actions in the relaxed plan because the actions are selected based on their supporting the goal or a subgoal (precondition). The classical relaxed plan heuristic h_{RP}^{SG} finds a set of (possibly interfering) actions to support the goal, using a subset of the initial layer literals as support. The relaxed plan is a levelled subgraph of the planning graph, of the form $\{ \mathcal{A}_0^{RP}, \mathcal{E}_0^{RP}, \mathcal{L}_1^{RP}, \dots, \mathcal{A}_{b-1}^{RP}, \mathcal{E}_{b-1}^{RP}, \mathcal{L}_b^{RP} \}$. Each of the layers contains a subset of the vertices in the corresponding layer of the planning graph.

More formally, we find the relaxed plan to support the constituent $\hat{S} \in \hat{\xi}(BS_i)$ that is reached earliest in the graph (as found by the $h_{level}^{SG}(BS_i)$ heuristic in Appendix A). Briefly, $h_{level}^{SG}(BS_i)$ returns the first level b where a constituent of BS_i has all its literals are in \mathcal{L}_b and none are marked pair wise mutex. We start extraction at the level b , by defining \mathcal{L}_b^{RP} as the literals in the constituent used in the level heuristic. For each literal \mathcal{L}_b^{RP} , we select a supporting effect (ignoring mutexes) from \mathcal{E}_{b-1} to form the subset \mathcal{E}_{b-1}^{RP} . We prefer persistence of literals to effects in supporting literals. Once a supporting set of effects is found, we create \mathcal{A}_{b-1}^{RP} as all actions with an effect in \mathcal{E}_{b-1}^{RP} . Then the needed preconditions for the actions and antecedents for chosen effects in \mathcal{A}_{b-1}^{RP} and \mathcal{E}_{b-1}^{RP} are added to the list of literals to support from \mathcal{L}_{b-2}^{RP} . Then, we find support with effects at level $b - 2$. The algorithm repeats until we find the needed actions from \mathcal{A}_0 . A relaxed plan’s value is the sum of the number of actions in each action layer. A literal persistence, denoted by a subscript “p”, is treated as an action in the planning graph, but in relaxed plan we do not include it in the final computation of $|\mathcal{A}_j^{RP}|$. The single graph relaxed plan heuristic is computed as,

$$h_{RP}^{SG}(BS_i) = \sum_{j=0}^{b-1} |\mathcal{A}_j^{RP}|$$

While we are using all literals in the source belief state as the initial layer of the planning graph, we are only extracting the relaxed plan for the literals appearing in a constituent of the destination belief state. Thus, we approximate two states and use the single planning graph relaxed plan to measure their state to state distance.

For the CBTC problem we find a relaxed plan, as shown in Figure 5 as the bolded edges and nodes. We found that \neg arm and \neg clog are non mutex at level two, then use persistence

to support $\neg\text{clog}$ and DunkP1 to support $\neg\text{arm}$ in \mathcal{L}_2^{RP} . In \mathcal{L}_1^{RP} we use persistence for inP1, and Flush for $\neg\text{clog}$. Thus, $h_{RP}^{SG}(BS_G) = 2$ because the relaxed plan is:

$$\begin{aligned} \mathcal{A}_0^{RP} &= \{\text{inP1}_p, \text{Flush}\}, \\ \mathcal{E}_0^{RP} &= \{\varphi^0(\text{inP1}_p), \varphi^0(\text{Flush})\}, \\ \mathcal{L}_1^{RP} &= \{\text{inP1}, \neg\text{clog}\}, \\ \mathcal{A}_1^{RP} &= \{\neg\text{clog}_p, \text{DunkP1}\}, \\ \mathcal{E}_1^{RP} &= \{\varphi^0(\neg\text{clog}_p), \varphi^1(\text{DunkP1})\}, \\ \mathcal{L}_2^{RP} &= \{\neg\text{arm}, \neg\text{clog}\} \end{aligned}$$

The relaxed plan does not use both DunkP2 and DunkP1 to support $\neg\text{arm}$. Notice that $\neg\text{arm}$ is not supported in all worlds (i.e. it is not supported when the state where inP2 holds is our initial state). Our initial layer threw away knowledge of inP1 and inP2 holding in different worlds, and the relaxed plan extraction ignored that $\neg\text{arm}$ needs to be supported in all worlds. A single, unmodified classical planning graph cannot capture support from all possible worlds – hence there is no explicit aggregation over distance measures for states. Rather we approximate belief states with states, so we can get a distance measure.

Positive Interaction Aggregation: Individual state to state distances can not be computed on the single planning graph because all notion of which state in BS_P is used for the estimate is lost we form the initial layer.

Independence Aggregation: See Positive Interaction Aggregation.

Overlap Aggregation: See Positive Interaction Aggregation.

4.3 Multiple Graph Heuristics (MG)

Single graph heuristics are usually uninformed because the projected belief state BS_P 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 the CBTC problem where the projected belief state is BS_I and we are using a single graph. If DunkP1 were the only action we would say that $\neg\text{arm}$ and $\neg\text{clog}$ can be reached at a cost of two, but in fact the cost is infinite (since there is no DunkP2 to support $\neg\text{arm}$ from all possible worlds), and there is no strong plan.

To account for lack of support in all possible worlds and sharpen the heuristic estimate, a set of planning graphs Γ is considered. Each $\gamma \in \Gamma$ is a single graph, like previously discussed. However, we can prevent loss of possible world distinctions by constructing the initial layer \mathcal{L}_0^γ of each graph γ with a different state $S \in \xi(BS_P)$. With multiple graphs, the heuristic value of a belief state is computed in terms of all the graphs. Unlike single graphs, we can compute different world aggregation measures with the multiple planning graphs. The idea is to compute a heuristic on each graph and then combine the heuristics to get a belief state distance.

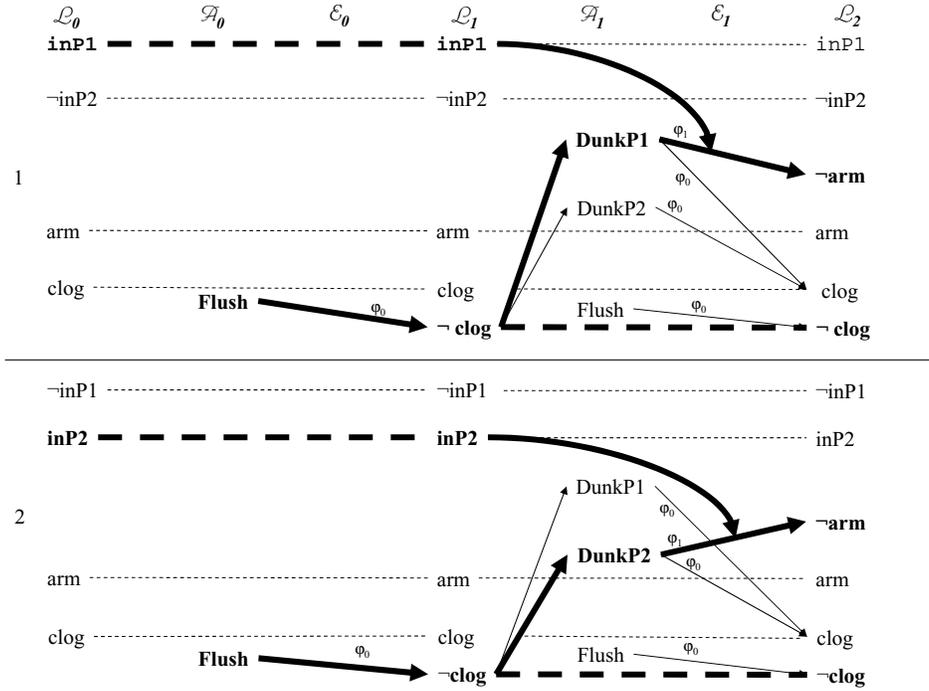


Figure 6: Multiple planning graphs for CBTC, with relaxed plan components bolded. Mutexes omitted.

These multiple graphs are similar to the graphs used by CGP [Smith and Weld, 1998], but lack the more general cross-world mutexes. Mutexes are only computed within each graph, i.e. only same-world mutexes are computed.

To illustrate using multiple graphs, consider regression search in CBTC. We will build two graphs (Figure 6) for the projected BS_I . They have the respective initial literal layers:

$$\mathcal{L}_0^1 = \{\text{arm}, \text{clog}, \text{inP1}, \neg\text{inP2}\}$$

$$\mathcal{L}_0^2 = \{\text{arm}, \text{clog}, \neg\text{inP2}, \text{inP2}\}$$

In the graph for the first possible world, S_1 , $\neg\text{arm}$ comes in only through DunkP1 at level 2. In the graph for the second world, S_2 , $\neg\text{arm}$ comes in only through DunkP2 at level 2. Thus, the multiple graphs show which actions in the different worlds contribute to support the same fact.

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

No Aggregation: A single planning graph is sufficient if there is no state aggregation being measured.

Positive Interaction Aggregation: Similar to GPT, we can use the worst-case world to represent the cost of the belief state by using the h_{m-RP}^{MG} heuristic. With this heuristic we account for the number of actions used in a given world, but assume positive interaction in all worlds.

The h_{m-RP}^{MG} heuristic is computed by finding a relaxed plan RP_γ on each planning graph $\gamma \in \Gamma$, exactly as done on the single graph with h_{RP}^{SG} . The difference is that unlike the single graph relaxed plan, the initial levels of the planning graphs are states, so each relaxed plan will reflect all the support needed in the world corresponding to γ .

Formally

$$h_{m-RP}^{MG}(BS_i) = \max_{\gamma \in \Gamma} \left(\sum_{j=0}^{b_\gamma-1} | \mathcal{A}_j^{RP_\gamma} | \right)$$

where b_γ is the level of γ where a constituent of BS_G was first reachable.

For CBTC, computing $h_{m-RP}^{MG}(BS_G)$ (Figure 6) finds $RP_1 =$

$$\begin{aligned} \mathcal{A}_0^{RP_1} &= \{\text{inP1}_p, \text{Flush}\}, \\ \mathcal{E}_0^{RP_1} &= \{\varphi^0(\text{inP1}_p), \varphi^0(\text{Flush})\}, \\ \mathcal{L}_1^{RP_1} &= \{\text{inP1}, \neg\text{clog}\}, \\ \mathcal{A}_1^{RP_1} &= \{\neg\text{clog}_p, \text{DunkP1}\}, \\ \mathcal{E}_1^{RP_1} &= \{\varphi^0(\neg\text{clog}_p), \varphi^1(\text{DunkP1})\}, \\ \mathcal{L}_2^{RP_1} &= \{\neg\text{arm}, \neg\text{clog}\} \end{aligned}$$

and $RP_2 =$

$$\begin{aligned} \mathcal{A}_0^{RP_2} &= \{\text{inP2}_p, \text{Flush}\}, \\ \mathcal{E}_0^{RP_2} &= \{\varphi^0(\text{inP2}_p), \varphi^0(\text{Flush})\}, \\ \mathcal{L}_1^{RP_2} &= \{\text{inP2}, \neg\text{clog}\}, \\ \mathcal{A}_1^{RP_2} &= \{\neg\text{clog}_p, \text{DunkP2}\}, \\ \mathcal{E}_1^{RP_2} &= \{\varphi^0(\neg\text{clog}_p), \varphi^1(\text{DunkP2})\}, \\ \mathcal{L}_2^{RP_2} &= \{\neg\text{arm}, \neg\text{clog}\} \end{aligned}$$

Taking the max of the two relaxed plan values gives $h_{m-RP}^{MG}(BS_G) = 2$.

Independence Aggregation: We can use the h_{s-RP}^{MG} heuristic to assume independence among the worlds in our belief. We extract relaxed plans exactly as described in the previous heuristic and simply use a summation rather than maximization of the relaxed plan costs. Formally

$$h_{s-RP}^{MG}(BS_i) = \sum_{\gamma \in \Gamma} \left(\sum_{j=0}^{b_\gamma-1} | \mathcal{A}_j^{RP_\gamma} | \right)$$

where b_γ is the level of γ where a constituent of BS_G was first reachable.

For CBTC, if computing $h_{s-RP}^{MG}(BS_G)$ (Figure 6), we find the same relaxed plans as in $h_{m-RP}^{MG}(BS_G)$, but sum their values to get $2 + 2 = 4$ as our heuristic.

State Overlap Aggregation: We notice that in the two previous heuristics we are taking a maximization and not accounting for some actions, or taking a summation and accounting for extra actions. We present the h_{RPU}^{MG} heuristic to balance the measure between positive interaction and independence of worlds. Examining the relaxed plans computed by the two previous heuristics for the CBTC example, we see that the relaxed plans extracted from each graph have some overlap. Notice, that both $\mathcal{A}_0^{RP_1}$ and $\mathcal{A}_0^{RP_2}$ contain a Flush action irrespective of which package the bomb is in – showing some positive interaction. Also, $\mathcal{A}_1^{RP_1}$ contains DunkP1, and $\mathcal{A}_1^{RP_2}$ contains DunkP2 – showing some independence. If we take the layer-wise union of the two relaxed plans, we would get a unioned relaxed plan $RP_U =$

$$\begin{aligned} \mathcal{A}_0^{RP_U} &= \{\text{inP1}_p, \text{Flush}\}, \\ \mathcal{E}_0^{RP_U} &= \{\varphi^0(\text{inP1}_p), \varphi^0(\text{inP2}_p), \varphi^0(\text{Flush})\}, \\ \mathcal{L}_1^{RP_U} &= \{\text{inP1}, \text{inP2}, \neg\text{clog}\}, \\ \mathcal{A}_1^{RP_U} &= \{\neg\text{clog}_p, \text{DunkP1}, \text{DunkP2}\}, \\ \mathcal{E}_1^{RP_U} &= \{\varphi^0(\neg\text{clog}_p), \varphi^1(\text{DunkP1}), \varphi^1(\text{DunkP2})\}, \\ \mathcal{L}_2^{RP_U} &= \{\neg\text{arm}, \neg\text{clog}\} \end{aligned}$$

This relaxed plans accounts for the actions that are the same between possible worlds and the actions that differ.

In order to get the union relaxed plan, we extract relaxed plans from each $\gamma \in \Gamma$, as in the two previous heuristics. Then, starting from the last level and repeating for each level, 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 layer of each relaxed plan to create the last layer of the RP_U relaxed plan, then the second to last layer for each relaxed plan is unioned and so on. The sum of the numbers of actions of each action level in the unioned relaxed plan is used as the heuristic value. Formally

$$h_{RPU}^{MG}(BS_i) = \sum_{j=0}^{b-1} |\mathcal{A}_j^{RP_U}|$$

where b is the greatest level b_γ where a constituent of BS_G was first reachable.

For CBTC, we just found RP_U , so counting the number of actions gives us a heuristic value of $h_{RPU}^{MG}(BS_G) = 3$.

4.4 Labelled Uncertainty Graph Heuristics (LUG)

The multiple graph technique has the advantage of heuristics that can aggregate the costs of multiple worlds, but the disadvantages of computing some redundant information in different graphs (c.f. Figure 6) and using every graph to compute heuristics (c.f h_{RPU}^{MG}). Our next approach addresses these limitations by condensing the multiple planning graphs to a single planning graph, called a labelled uncertainty graph (LUG). The idea is to

implicitly represent multiple planning graphs by collapsing the graph connectivity into one planning graph, but use annotations, called labels (ℓ), to retain information about multiple worlds. While we could construct the *LUG* by generating each of the multiple graphs and taking their union, instead we define a direct construction procedure. We start similar to the single planning graph by constructing an initial layer of all literals in our source belief. The difference with the *LUG* is that we can prevent loss of information about multiple worlds by recording in a label for each literal which of the worlds is relevant. As we will discuss, we use a few simple techniques to propagate the labels through actions and effects to label subsequent literal layers. Label propagation relies on expressing labels as propositional formulas and using standard propositional logic operations. The end product is single planning graph with labels on all graph elements; labels denote in which of the explicit multiple graphs (if we were to build them) a graph element is reached.

We are trading planning graph structure space for label storage space. Our choice of BDDs to represent labels helps lower the storage requirements on labels. 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. 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. A nice feature of BDDs in the package we use [Brace *et al.*, 1990] is that they efficiently represent many BDDs in a shared BDD that leverages common substructure. Hence, in practice the *LUG* contains the same information as *MG* with much lower construction and usage costs.

In this section we present how to construct the *LUG* without mutexes, then describe how to introduce mutexes, and finally discuss how to extract relaxed plans.

4.4.1 LABEL PROPAGATION

The *LUG* is based on the *IPP* [Koehler, 1999] planning graph, where a planning graph level has an the action layer, effect layer, and literal layer. The extension is to add labels to the elements of the action \mathcal{A} , effect relation \mathcal{E} , and literal \mathcal{L} layers. We denote the label of a graph element in level k as $\ell_k(\cdot)$. We can build the *LUG* for any belief state BS_P . We illustrate the case where $BS_P = BS_I$ from the CBTC example.

A label is a formula describing a set of states from which a graph element is (optimistically) *reachable*. We say a literal l is reachable from a set of states, described by BS_i , after k steps, if $BS_i \models BS_P$ and $BS_i \models \ell_k(l)$. There are also times where we would like to know if a propositional formula (e.g. an execution precondition) is reachable from BS_i . For this, we introduce notation for extended labels $\ell_k^*(f)$ of formulae f . We say that any propositional formula f is reachable from BS_i after k steps if $BS_i \models \ell_k^*(f)$. Since we only

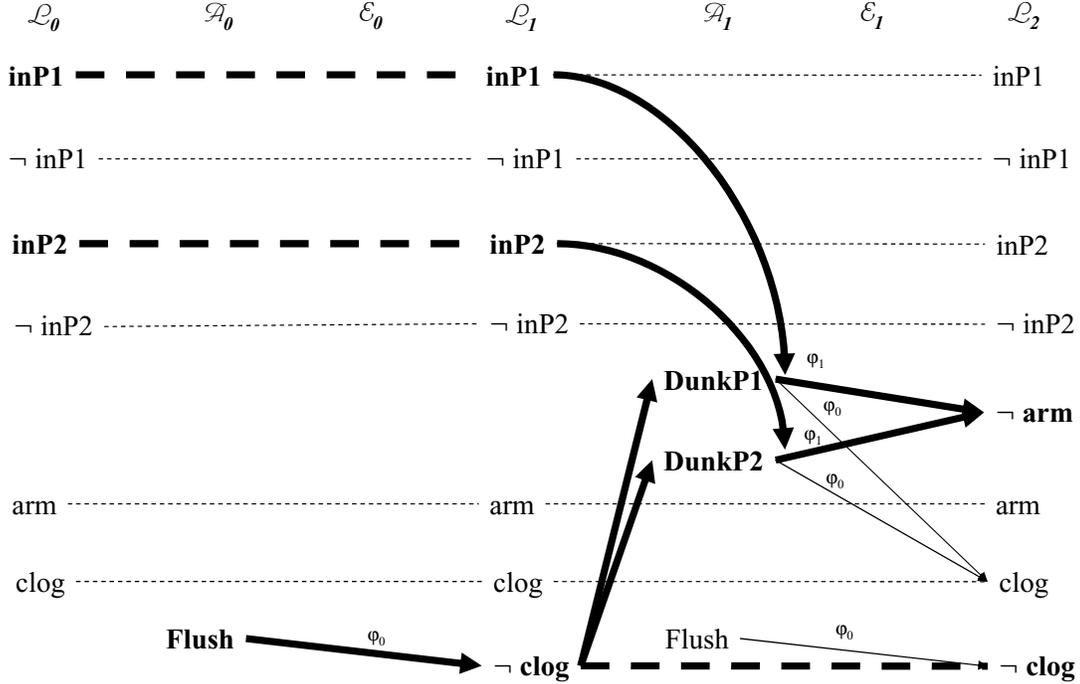


Figure 7: *LUG* for CBTC, with no mutexes. The relaxed plan for h_{RP}^{LUG} is shown in bold.

have labels for literals, we substitute the labels of literals for the literals in a formula to get the extended label of the formula. The extended label of a propositional formula f at level k , is defined:

$$\begin{aligned}
 \ell_k^*(f \wedge f') &= \ell_k^*(f) \wedge \ell_k^*(f'), \\
 \ell_k^*(f \vee f') &= \ell_k^*(f) \vee \ell_k^*(f'), \\
 \ell_k^*(\neg(f \wedge f')) &= \ell_k^*(\neg f \vee \neg f'), \\
 \ell_k^*(\neg(f \vee f')) &= \ell_k^*(\neg f \wedge \neg f'), \\
 \ell_k^*(\top) &= BS_P, \\
 \ell_k^*(\perp) &= \perp, \\
 \ell_k^*(l) &= \ell_k(l)
 \end{aligned}$$

If we build the *LUG* for CBTC, using $BS_P = BS_I$, we could say that BS_G is reachable from BS_I after two steps if the literal layer \mathcal{L}_2 contains $\neg \text{arm}$ and $\neg \text{clog}$, and $BS_I \models \ell_2^*(\neg \text{arm} \wedge \neg \text{clog})$, meaning that the models of worlds where $\neg \text{arm} \wedge \neg \text{clog}$ holds after two steps are a superset of the worlds in our current belief. To capture this intuition in labelling, the propagation of labels must reflect that (i) actions and effects are applicable in the possible worlds for which their conditions are reachable and (ii) a literal is reachable in all

possible worlds where it is given as an effect. The construction involves defining an initial literal layer and an inductive step to define a level.

Initial Literal Layer: The LUG has an initial layer consisting of every literal with a non false (\perp) label. In the initial layer the label $\ell_0(l)$ of each literal l is identical to $l \wedge BS_P$, representing the states of BS_P in which l holds.

The labels for the initial layer literals are propagated through actions and effects to label the next literal layer, as we will describe shortly. We continue propagation until no label of any literal changes between layers, a condition referred to as level off.

The LUG for CBTC, Figure 7 (shown in without labels), using $BS_P=BS_I$ has the initial literal layer:

$$\begin{aligned} \mathcal{L}_0 = \{ & \ell_0(\neg \text{inP2}) = \ell_0(\text{inP1}) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg \text{inP2}), \\ & \ell_0(\neg \text{inP1}) = \ell_0(\text{inP2}) = (\text{arm} \wedge \text{clog} \wedge \neg \text{inP1} \wedge \text{inP2}), \\ & \ell_0(\text{clog}) = \ell_0(\text{arm}) = BS_P \} \end{aligned}$$

Notice that inP1 and inP2 have labels indicating the respective initial states in which they hold, and clog and arm have BS_P as their label because they hold in all states in BS_P .

Action Layer: Once the previous literal layer \mathcal{L}_k is computed, we find the action layer \mathcal{A}_k . \mathcal{A}_k contains causative actions from the action set A , plus literal persistence. Persistence for a literal l , denoted by l_p , is represented as an action where $\rho^e(l_p) = \varepsilon^0(l_p) = l$. An action is included in \mathcal{A}_k if its label is non empty (i.e. $\ell_k(a) \neq \perp$). The label of an action at level k , is equivalent to the extended label of its execution precondition:

$$\ell_k(a) = \ell_k^*(\rho^e(a))$$

The zeroth action layer for CBTC, is:

$$\begin{aligned} \mathcal{A}_0 = \{ & \ell_0(\text{Flush}) = BS_P, \\ & \ell_0(\neg \text{inP2}_p) = \ell_0(\text{inP1}_p) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg \text{inP2}), \\ & \ell_0(\neg \text{inP1}_p) = \ell_0(\text{inP2}_p) = (\text{arm} \wedge \text{clog} \wedge \neg \text{inP1} \wedge \text{inP2}), \\ & \ell_0(\text{clog}_p) = \ell_0(\text{arm}_p) = BS_P \} \end{aligned}$$

The literal persistences have labels identical to the label of the corresponding literal from the previous literal layer. The Flush action has BS_P as its label because it is always applicable.

Effect Layer: The effect layer \mathcal{E}_k depends both on the literal layer \mathcal{L}_k and action layer \mathcal{A}_k . \mathcal{E}_k contains an effect $\varphi^j(a)$ if the effect has a non empty label (i.e. $\ell_k(\varphi^j(a)) \neq \perp$). Because both the action and an effect must be applicable in the same work, the label of the effect at level k is the conjunction of the label of the associated action with the extended label of the antecedent:

$$\ell_k(\varphi^j) = \ell_k(a) \wedge \ell_k^*(\rho^j(a))$$

The zeroth effect layer for CBTC, is:

$$\begin{aligned} \mathcal{E}_0 = \{ & \ell_0(\varphi^0(\text{Flush})) = BS_P \\ & \ell_0(\varphi^0(\neg\text{inP2}_p)) = \ell_0(\varphi^0(\text{inP1}_p)) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_0(\varphi^0(\neg\text{inP1}_p)) = \ell_0(\varphi^0(\text{inP2}_p)) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_0(\varphi^0(\text{clog}_p)) = \ell_0(\varphi^0(\text{arm}_p)) = BS_P \} \end{aligned}$$

Again, like the action layer, the unconditional effect of each literal persistence has a label identical to the corresponding literal in the previous literal layer. The unconditional effect of Flush has a label identical to the label of Flush.

Literal Layer: The literal layer \mathcal{L}_k depends on the previous effect layer \mathcal{E}_{k-1} , and contains only literals with non empty labels (i.e. $\ell_k(l) \neq \perp$). An effect $\varphi^j(a) \in \mathcal{E}_{k-1}$ contributes to the label of a literal l when the effect consequent contains the literal l . The label of a literal is the disjunction of the labels of each effect from the previous effect layer that gives the literal:

$$\ell_k(l) = \bigvee_{\substack{\varphi^j(a): l \in \varepsilon^j(a), \\ \varphi^j(a) \in \mathcal{E}_{k-1}}} \ell_{k-1}(\varphi^j(a))$$

The first literal layer for CBTC is:

$$\begin{aligned} \mathcal{L}_1 = \{ & \ell_1(\neg\text{inP2}) = \ell_1(\text{inP1}) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_1(\neg\text{inP1}) = \ell_1(\text{inP2}) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_1(\neg\text{clog}) = \ell_1(\text{clog}) = \ell_1(\text{arm}) = BS_P \} \end{aligned}$$

This literal layer is identical the initial literal layer, except that $\neg\text{clog}$ goes from having a false label \perp (i.e. not existing in the layer) to having the label BS_P .

We continue to the level one action layer:

$$\begin{aligned} \mathcal{A}_1 = \{ & \ell_1(\text{DunkP1}) = \ell_1(\text{DunkP2}) = \ell_1(\text{Flush}) = BS_P, \\ & \ell_1(\neg\text{inP2}_p) = \ell_1(\text{inP1}_p) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_1(\neg\text{inP1}_p) = \ell_1(\text{inP2}_p) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_1(\text{clog}_p) = \ell_1(\text{arm}_p) = BS_P \} \end{aligned}$$

This action layer is identical to the level zero action layer, but adds both Dunk actions because they are now executable due to $\neg\text{clog}$, and the persistence for $\neg\text{clog}$. Each Dunk action gets a label identical to their execution precondition label.

Followed by the level one effect layer:

$$\begin{aligned} \mathcal{E}_1 = \{ & \ell_1(\varphi^0(\text{DunkP1})) = \ell_1(\varphi^0(\text{DunkP2})) = \ell_1(\varphi^0(\text{Flush})) = BS_P \\ & \ell_1(\varphi^1(\neg\text{DunkP1})) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_1(\varphi^1(\neg\text{DunkP2})) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_1(\varphi^0(\neg\text{inP2}_p)) = \ell_1(\varphi^0(\text{inP1}_p)) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_1(\varphi^0(\neg\text{inP1}_p)) = \ell_1(\varphi^0(\text{inP2}_p)) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_1(\varphi^0(\text{clog}_p)) = \ell_1(\varphi^0(\text{arm}_p)) = BS_P \} \end{aligned}$$

The conditional effects of the Dunk actions in CBTC (Figure 7) have labels that indicate the possible worlds in which they will give $\neg\text{arm}$ because their antecedents do not hold in all possible worlds. For example, the conditional effect $\varphi^1(\text{DunkP1})$ has the label found by taking the conjunction of the action's label BS_P with the antecedent's label $(\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2})$.

Finally, the level two literal layer:

$$\begin{aligned} \mathcal{L}_2 = \{ & \ell_2(\neg\text{inP2}) = \ell_2(\text{inP1}) = (\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\ & \ell_2(\neg\text{inP1}) = \ell_2(\text{inP2}) = (\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\ & \ell_2(\neg\text{clog}) = \ell_2(\text{clog}) = \ell_2(\text{arm}) = \ell_2(\neg\text{arm}) = BS_P \} \end{aligned}$$

The labels of the literals for level 2 of CBTC, Figure 7, indicate that $\neg\text{arm}$ is reachable because its label is entailed by BS_I . The label of $\neg\text{arm}$ is found by taking the disjunction of the labels of effects that give it, namely, $(\text{arm} \wedge \text{clog} \wedge \text{inP1} \wedge \neg\text{inP2})$ from the conditional effect of DunkP1 and $(\text{arm} \wedge \text{clog} \wedge \neg\text{inP1} \wedge \text{inP2})$ from the conditional effect of DunkP2, which reduces to BS_P . Construction could stop here because BS_P entails the label of the goal $\ell_k^*(\neg\text{arm} \wedge \neg\text{clog}) = \ell_k(\neg\text{arm}) \wedge \ell_k(\neg\text{clog}) = BS_P \wedge BS_P = BS_P$. However, level off occurs at the next level because there is no change in the labels of the literals.

When level off occurs at level three in our example, we can say that for any BS_i , where $BS_i \models BS_P$, that a formula f is reachable in k steps if $\exists_k BS_i \models \ell_k^*(f)$. If no such level k exists, then f is not reachable from BS_i . If there is some level k , where f is reachable from BS_i , then the first such k is a lower bound on the number of parallel plan steps needed to reach f from BS_i . This lower bound is similar to the classical planning max heuristic [Nguyen *et al.*, 2002]. We can provide a more informed heuristic by extracting a relaxed plan to support f with respect to BS_i , described shortly.

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 in a single possible world because same-world mutexes are more effective as well as relatively easy to understand. We extend the mutex propagation that was used in the multiple graphs so that the mutexes are on one planning graph. The savings of computing on the *LUG*

instead of multiple graphs is that we can reduce computation when a mutex exists in several worlds. In Appendix B we describe how to handle cross-world mutexes, despite their lack of effectiveness in the experiments we conducted. Cross world mutexes extend the *LUG* to compute the same set of mutexes found by CGP [Smith and Weld, 1998].

Same-world mutexes can be represented with a single label, $\hat{\ell}_k(x_1, x_2)$, between two elements (actions, effect, or literals). The mutex holds between elements x_1 and x_2 in all worlds S where $S \models \hat{\ell}_k(x_1, x_2)$. If the elements are not mutex in any world, we can assume the label of a mutex between them is empty \perp . We discuss how the labelled mutexes are discovered and propagated for actions, effect relations, and literals.

When using mutexes, we can refine what it means for a formula f to be reachable from a set of worlds BS_P . We must ensure that for every state in the source belief BS_P , there exists a state of f that is reachable. A state S' of f is reachable from a state S of BS_P when there are no two literals in S' that are mutex in world S and $BS_P \models \ell_k^*(S)$.

In each of the action, effect, and literal layers there are multiple ways for the same pair of elements to become mutex (e.g. interference or competing needs). Thus, the mutex label for a pair is disjunction of all labelled mutexes found for the pair by some means.

Action Mutexes \hat{A}_k : The same world action mutexes at a level k 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** Two actions a, a' interfere if (1) the unconditional effect consequent $\varepsilon^0(a)$ of one is inconsistent with either the (a) execution precondition $\rho^e(a')$ or (b) unconditional effect consequent $\varepsilon^0(a')$ of the other, or (2) the execution preconditions $\rho^e(a), \rho^e(a')$ are inconsistent. The mutex will exist in all possible world projections $\hat{\ell}_k(a, a') = BS_P$. Formally, a and a' interfere if:

$$(1a) \ \varepsilon^0(a) \wedge \rho^e(a') = \perp$$

$$(1b) \ \varepsilon^0(a) \wedge \varepsilon^0(a') = \perp$$

$$(2) \ \rho^e(a) \wedge \rho^e(a') = \perp$$

- **Competing Needs** Two actions a, a' have competing needs in a world when a pair of literals from their execution preconditions are mutex in the world. The worlds where a and a' are mutex because of competing needs are described by:

$$\perp \vee \bigvee_{l \in \rho^j(a), l' \in \rho^j(a')} \ell_k(l, l') \wedge \ell_k(a) \wedge \ell_k(a')$$

In the above formula we find that there are no worlds where the actions are mutex (\perp) or for all pairs of preconditions ($l \in \rho^e(a), l' \in \rho^e(a')$) the worlds where the

preconditions are mutex and both actions are reachable ($\ell_k(l, l') \wedge \ell_k(a) \wedge \ell_k(a')$), there exists a mutex between the actions.

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 associated action mutexes, interference, competing needs, or induced effects.

- **Mutex Actions** Two effects $\varphi^i(a) \in \Phi(a), \varphi^j(a') \in \Phi(a')$ of different actions a, a' are mutex in all worlds where their associated actions are mutex, $\hat{\ell}_k(a, a')$.
- **Interference** Like actions, two effects interfere if (1) the consequent of one effect $\varepsilon^i(a)$ and (a) antecedent $\rho^j(a')$ or (b) consequent $\varepsilon^j(a')$ of the other are inconsistent, or (2) the antecedents $\rho^i(a), \rho^j(a')$ are inconsistent. The mutex will exist in all possible world projections, so the label of the mutex is $\hat{\ell}_k(\varphi^i(a), \varphi^j(a')) = BS_P$. Formally, $\varphi^i(a)$ and $\varphi^j(a')$ interfere if:

$$(1a) \ \varepsilon^i(a) \wedge \rho^j(a') \models \perp$$

$$(1b) \ \varepsilon^i(a) \wedge \varepsilon^j(a') \models \perp$$

$$(2) \ \rho^i(a) \wedge \rho^j(a') \models \perp$$

- **Competing Needs** Like actions, two effects have competing needs in a world when a pair of literals from their antecedents are mutex in a world. The possibly empty label of a mutex between $\varphi^i(a)$ and $\varphi^j(a')$ is in worlds:

$$\perp \vee \bigvee_{l \in \rho^i(a), l' \in \rho^j(a')} \ell_k(l, l') \wedge \ell_k(\varphi^i(a)) \wedge \ell_k(\varphi^j(a'))$$

In the above formula we find that there are no worlds where the effects are mutex (\perp) or for all pairs of preconditions ($l \in \rho^i(a), l' \in \rho^j(a')$) the worlds where the preconditions are mutex and both actions are reachable ($\ell_k(l, l') \wedge \ell_k(\varphi^i(a)) \wedge \ell_k(\varphi^j(a'))$) there exists a mutex between the actions.

- **Induced** An induced effect of an effect is an effect of the same action that may execute at the same time. An effect is induced by another in the possible worlds where they are both reachable. For example, the conditional effect of an action always induces the unconditional effect of the action. Two effects of different actions are mutex because of induced effects when an action has two effects where one is (i) mutex with the effect of another action, and (ii) the effects of the action can be executed together (i.e. induce each other).

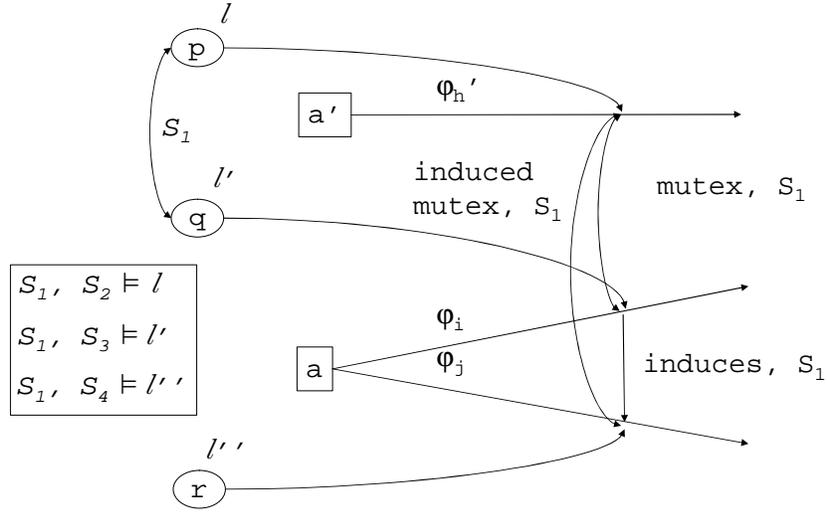


Figure 8: Effect $\varphi^i(a)$ induces effect $\varphi^j(a)$. $\varphi^i(a)$ is mutex with $\varphi^h(a')$, so $\varphi^j(a)$ is induced mutex with $\varphi^h(a')$.

The induced 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 $\varphi^i(a)$ is mutex with $\varphi^h(a')$ in the possible worlds described by $\hat{\ell}_k(\varphi^i(a), \varphi^h(a'))$, and $\varphi^i(a)$ induces effect $\varphi^j(a)$ in the possible worlds described by $\ell_k(\varphi^i(a)) \wedge \ell_k(\varphi^j(a))$, then there is an induced mutex between $\varphi^j(a)$ and $\varphi^h(a')$ for all possible worlds specified by $\ell_k(\varphi^i(a)) \wedge \ell_k(\varphi^j(a)) \wedge \hat{\ell}_k(\varphi^i(a), \varphi^h(a'))$ (see Figure 8). For additional 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 8. 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 $\varphi^i(a)$ of a induces $\varphi^j(a)$ in possible world S_1 , $\varphi^i(a)$ is mutex with effect $\varphi^h(a')$ in possible world S_1 because of the mutex between p and q , and $\varphi^j(a)$ becomes induced mutex with $\varphi^h(a')$ 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 only reason for mutex literals is inconsistent support.

- **Inconsistent Support** Two literals have inconsistent support in a possible world at level k when there are no two non-mutex effects that support both literals in the world. The label of the literal mutex at level k is a disjunction of all worlds where they have inconsistent support. The worlds for a possibly empty inconsistent support mutex between l and l' are:

$$\perp \vee \bigvee_{\substack{S: \forall \varphi^i(a), \varphi^j(a') \in \mathcal{E}_{k-1}, \\ \text{where } l \in \varepsilon^i(a), l' \in \varepsilon^j(a'), \\ S \models \hat{\mathcal{L}}_{k-1}(\varphi^i(a), \varphi^j(a'))}} S$$

The meaning of the above formula is that the literals are mutex in all worlds S where all pairs of effects that support the literals in S are mutex in S .

4.4.2 LUG HEURISTICS

The heuristics computed on the *LUG* can capture measures similar to the *MG* heuristics, but there exists a new opportunity to leverage labels to make some more efficient.

No Aggregation: A single planning is sufficient if there is no state aggregation being measured.

Positive Interaction Aggregation: Unlike *MG* heuristics, we do not compute positive interaction based relaxed plans on the *LUG*. The *MG* approach to measure positive interaction across each state in a belief is using multiple relaxed plans and taking their max value. To get the same measure on the *LUG* we would still need to extract multiple relaxed plans, the situation we are trying to avoid by using the *LUG*. While the graph construction overhead may be lowered by using the *LUG*, the heuristic computation could take too long. Hence, we do not compute relaxed plans on the *LUG* to measure positive interaction alone, but we do compute relaxed plans that measure overlap (which measures positive interaction).

Independence Aggregation: Like positive interaction aggregation, we need a relaxed plan for every state in the projected belief to find the sum of the costs. Hence, we do not use any relaxed plans that assume world independence.

State Overlap Aggregation: Relaxed plans extracted from the *LUG* to get the h_{RP}^{LUG} heuristic resemble the unioned relaxed plan in the h_{RPU}^{MG} heuristic. Recall that the h_{RPU}^{MG} 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 have positive interaction in multiple worlds only once and accounts for independent actions that are used in subsets of the possible worlds. The advantage of h_{RP}^{LUG} is that we find these actions with a single pass on one planning graph.

We are trading the cost of computing multiple relaxed plans for the cost of manipulating *LUG* labels to determine what lines of causal support are used in what worlds. In the relaxed plan we want to support the goal with every state in BS_P , but in doing so we need to track which states in BS_P use which paths in the planning graph. A subgoal may have several different (and possibly overlapping) paths from the worlds in BS_P .

The *LUG* relaxed plans are labelled subgraphs of a *LUG*, represented as a set of layers: $\{\mathcal{A}_0^{RP}, \mathcal{E}_0^{RP}, \mathcal{L}_1^{RP}, \dots, \mathcal{A}_{b-1}^{RP}, \mathcal{E}_{b-1}^{RP}, \mathcal{L}_b^{RP}\}$, where \mathcal{A}_r^{RP} is a set of labelled actions, \mathcal{E}_r^{RP} is a set of labelled effects, and \mathcal{L}_{r+1}^{RP} is a set of labelled clauses. The components of the relaxed plan are labelled to indicate the worlds of BS_P where they are chosen for support. The relaxed plan is extracted from the level $b = h_{level}^{LUG}(BS_i)$ (i.e., the first level where BS_i is reachable, also described in Appendix A).

Please note that we are extracting the relaxed plan for BS_i in terms of clauses, and not literals, which is different that the *SG* and *MG* versions of relaxed plans. Previously we found the constituent of BS_i that was first reached on a planning graph and now we do not commit to any one constituent. Our rationale is that we were possibly using different constituents in each of the multiple graphs, and in this condensed version of the multiple graphs we still want to be able to support different constituents of the BS_i in different worlds. We could also use the constituent representation of BS_i in defining the layers of the relaxed plan, but choose the clausal representation of BS_i instead because we know that we have to support each clause, however with constituents we know we only need to support one (but we don't need to know which one).

The relaxed plan, shown in bold in Figure 7, for BS_I to reach BS_G in CBTC is:

$$\begin{aligned}
 \{\mathcal{A}_0^{RP} &= \{\ell_0^{RP}(\text{inP1}_p) = (\text{arm} \wedge \neg\text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\
 &\quad \ell_0^{RP}(\text{inP2}_p) = (\text{arm} \wedge \neg\text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\
 &\quad \ell_0^{RP}(\text{Flush}) = BS_P\}, \\
 \mathcal{E}_0^{RP} &= \{\ell_0^{RP}(\varphi^0(\text{inP1}_p)) = (\text{arm} \wedge \neg\text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\
 &\quad \ell_0^{RP}(\varphi^0(\text{inP2}_p)) = (\text{arm} \wedge \neg\text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\
 &\quad \ell_0^{RP}(\varphi^0(\text{Flush})) = BS_P\}, \\
 \mathcal{L}_1^{RP} &= \{\ell_1^{RP}(\text{inP1}) = (\text{arm} \wedge \neg\text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\
 &\quad \ell_1^{RP}(\text{inP2}) = (\text{arm} \wedge \neg\text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\
 &\quad \ell_1^{RP}(\neg\text{clog}) = BS_P\}, \\
 \mathcal{A}_1^{RP} &= \{\ell_1^{RP}(\text{DunkP1}) = (\text{arm} \wedge \neg\text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\
 &\quad \ell_1^{RP}(\text{DunkP2}) = (\text{arm} \wedge \neg\text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\
 &\quad \ell_1^{RP}(\neg\text{clog}_p) = BS_P\}, \\
 \mathcal{E}_1^{RP} &= \{\ell_1^{RP}(\varphi^1(\text{DunkP1})) = (\text{arm} \wedge \neg\text{clog} \wedge \text{inP1} \wedge \neg\text{inP2}), \\
 &\quad \ell_1^{RP}(\varphi^1(\text{DunkP2})) = (\text{arm} \wedge \neg\text{clog} \wedge \neg\text{inP1} \wedge \text{inP2}), \\
 &\quad \ell_1^{RP}(\varphi^0(\neg\text{clog}_p)) = BS_P\}, \\
 \mathcal{L}_2^{RP} &= \{\ell_2^{RP}(\neg\text{arm}) = BS_P, \\
 &\quad \ell_2^{RP}(\neg\text{clog}) = BS_P\}
 \end{aligned}$$

We start by forming \mathcal{L}_2^{RP} with the clauses in $\kappa(BS_G)$, namely $\neg\text{arm}$ and $\neg\text{clog}$; we label the clauses with BS_P because they need to be supported by all states in our belief. Next, we support each clause in \mathcal{L}_2^{RP} with the relevant effects from \mathcal{E}_1 to form \mathcal{E}_1^{RP} . For $\neg\text{clog}$ we use persistence because it supports $\neg\text{clog}$ in all worlds described by BS_P (this is an example of positive interaction of worlds). For $\neg\text{arm}$ the relevant effects are the respective φ^1 from each Dunk action. We choose both effects to support $\neg\text{arm}$ because we need to support $\neg\text{arm}$ in all worlds of BS_P , and each effect gives support in only one world (this is an example of independence of worlds). We then insert the actions associated with each chosen effect into \mathcal{A}_1^{RP} with the appropriate label indicating the worlds it was *needed*, which in general is less worlds than it is *reachable* (i.e. it is always the case that $\ell_r^{RP}(\cdot) \models \ell_r(\cdot)$). Next we form \mathcal{L}_1^{RP} with the execution preconditions of actions in \mathcal{A}_1^{RP} and antecedents of effects in \mathcal{E}_1^{RP} , which are $\neg\text{clog}$, inP1 , and inP2 , labelled with all worlds where an action or effect needed them. In the same fashion as level two, we support the literals at level one, using persistence for inP1 and inP2 , and Flush for $\neg\text{clog}$. We stop here, because we have supported all clauses at level one.

For the general case, extraction starts at the level b where BS_i is first reachable from BS_P . The first relaxed plan layers we construct are $\mathcal{A}_{b-1}^{RP}, \mathcal{E}_{b-1}^{RP}, \mathcal{L}_b^{RP}$, where \mathcal{L}_b^{RP} contains all clauses $C \in \kappa(BS_i)$, labelled as $\ell_k^{RP}(C) = BS_P$.

For each level r , $1 \leq r \leq b$, we support each clause in \mathcal{L}_r^{RP} by choosing relevant effects from \mathcal{E}_{r-1} to form \mathcal{E}_{r-1}^{RP} . An effect $\varphi^j(a)$ is relevant if it is reachable in some of the worlds

where we need to support C (i.e. $\ell_{r-1}(\varphi^j(a)) \wedge \ell_r^{RP}(C) \neq \perp$) and the consequent gives a literal $l \in C$. For each clause, we have to choose enough supporting effects so that the chosen effect worlds are a superset of the worlds we need to support the clause, formally

$$\forall C \in \mathcal{L}_r^{RP} \ell_r^{RP}(C) \models \left(\bigvee_{\substack{\varphi^j(a): l \in \varepsilon^j(a), \\ l \in C, \\ \varphi^j(a) \in \mathcal{E}_{r-1}}} \ell_{r-1}^{RP}(\varphi^j(a)) \right)$$

We think of supporting a clause in a set of worlds as a set cover problem where effects cover subsets of worlds. Our algorithm to cover the worlds of a clause with worlds of effects is a variant of the well known greedy algorithm for set cover [Cormen *et al.*, 1990]. We first choose as many relevant persistence effects that can cover new worlds, then choose action effects that cover the most new worlds at each step. Each effect we choose for support is added to \mathcal{E}_{r-1}^{RP} and labelled with the new worlds it covered for C . Once all clauses in \mathcal{L}_r^{RP} are covered, we form the action layer \mathcal{A}_{r-1}^{RP} as all actions that have an effect in \mathcal{E}_{r-1}^{RP} . The actions in \mathcal{A}_{r-1}^{RP} are labelled to indicate all worlds where any of their effects were labelled in \mathcal{E}_{r-1}^{RP} .

We obtain the next subgoal layer, \mathcal{L}_{r-1}^{RP} , by adding literals from the execution preconditions of actions in \mathcal{A}_{r-1}^{RP} and antecedents of effects in \mathcal{E}_{r-1}^{RP} . Each literal $l \in \mathcal{L}_{r-1}^{RP}$ is labelled to indicate all worlds with which any action or effect, that required l , were labelled.

We support the literals in \mathcal{L}_{r-1}^{RP} in the same fashion as \mathcal{L}_r^{RP} . We continue to support literals with effects, insert actions, and insert action and effect preconditions until we have supported all literals in \mathcal{L}_1^{RP} .

Once we get a relaxed plan, the relaxed plan heuristic, $h_{RP}^{LUG}(BS_i)$, is the number of actions in each action layer, formally

$$h_{RP}^{LUG}(BS_i) = \sum_{i=0}^{b-1} |\mathcal{A}_i^{RP}|$$

Thus in our CBTC example we have $h_{RP}^{LUG}(BS_G) = 3$.

5. Empirical Evaluation

This section presents our implementation of the *CAltAlt* and *POND* planners and the results of our experimentation with the heuristics within them. All tests were run in Linux on a Pentium 4 2.66GHz w/ 1GB RAM. Both *CAltAlt* and *POND* used a heuristic weight of five in the, respective, A* and AO* searches. We also compare with the competing approaches (CGP, SGP, GPT v1.40, MBP v0.91, KACMBP, YKA, and CFF) on several

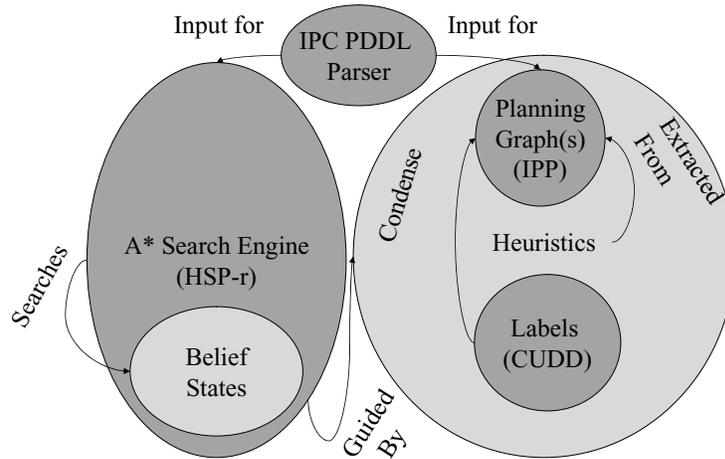


Figure 9: The implementation of *CAItAlt* relies on a regression search engine that searches over belief states. The search engine is guided by heuristics extracted from planning graphs.

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

5.1 Implementation

CAItAlt The implementation of *CAItAlt* uses several off-the-shelf planning software packages. Figure 9 shows a diagram of the system architecture. While *CAItAlt* extends the name of *AltAlt*, it relies on a limited subset of the implementation. The components of *CAItAlt* are the IPC parser for PDDL 2.1 (slightly extended to allow disjunction in the initial conditions), the HSP-r search engine [Bonet and Geffner, 1999], the IPP planning graph [Koehler *et al.*, 1997], and CUDD [Brace *et al.*, 1990] to implement the *LUG* labels. The custom parts of the implementation include the action representation, belief state representation and regression operator, and the heuristic calculation.

POND The implementation of *POND* is very similar to *CAItAlt* aside from the search engine and state representation. *POND* uses LAO* [Hansen and Zilberstein, 2001] source code from Eric Hansen to perform the search, and BDDs [Brace *et al.*, 1990] to represent belief states and actions. *POND* also uses the IPP source code for planning graphs.

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}	$2n-1$ { $n-1$ }
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 10: 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 conditional versions of domains is in braces; plan lengths are max conditional branch length.

5.2 Domains

Figure 10 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 Center, 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 [Long and Fox, 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 and three packages, each at one of two of the three post offices and having to reach different airports.

Conditional Problems For conditional planning we consider domains from the literature: bomb in the toilet with sensing BTS, and bomb in the toilet with clogging and sensing BTCS. We also extend the conformant Logistics and Rovers to include sensory 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.⁵

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

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

5.2.1 INTRA-PLANNER COMPARISON

We compare many techniques within *CAItAlt* and *POND* on our conformant planning domains, and in addition we test the heuristics in *POND* on the conditional domains. We proceed by showing how the heuristics perform in *CAItAlt* and then how various mutex computation schemes for the *LUG* can affect performance. Finally, we present how *POND* performs with the different heuristics in both conformant and conditional domains, then finish with a summary of important conclusions. Our performance metrics include the total planning time and the number of search nodes expanded. Additionally, when discussing mutexes we analyze planning graph construction time. We also compare the heuristic estimates in *POND* to the optimal plan length to gauge heuristic accuracy.

We only compute mutexes in the planning graphs for *CAItAlt* because the planning graph(s) are only built once in a search episode and mutexes help prune the inconsistent belief states encountered in regression search. We abstain from computing mutexes in *POND* because in progression we build new planning graphs for each search node and we want to keep graph computation time low.

CAItAlt: The results for *CAItAlt* in the conformant Rovers, Logistics, BT, and BTC domains, in terms of total time and number of expanded search nodes, are presented in Figure 11. We do not discuss the Ring and Cube Center domains for *CAItAlt* because due to implementation details the planner performs very poorly when domains have actions with several conditional effects and hence does not scale. We show the number of expanded nodes because it gives an indication of how well a heuristic guides the planner. The total time captures the amount of time computing the heuristic and searching. A high total time with a high number of search nodes indicates a poor heuristic, and a high total time and low number of search nodes indicates an expensive but informed heuristic.

We describe the results from left to right in Figure 11, comparing the different planning graph structures and relaxed plans computed on each planning graph. We start with the non-planning graph heuristics h_0 and h_{card} . As expected, h_0 , breadth-first search, does not perform well in a large portion of the problems, shown by the large number of search nodes and inability to scale to solve larger problems. We notice that with the h_{card} heuristic performance is very good in the BT and BTC problems (this confirms the results originally seen in [Bertoli *et al.*, 2001b]). However, h_{card} 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.

Next, for a single planning graph, *CAItAlt* does reasonably well with the h_{RP}^{SG} heuristic in the Rovers and Logistics domains, but fails to scale very well on the BT and BTC domains. Rovers and Logistics have comparatively fewer initial worlds than the BT and BTC problems. Moreover the deterministic plans, assuming each initial state is the real state, are somewhat similar for Rovers and Logistics, but mostly independent for BT and BTC.

Therefore, approximating a fully observable plan with the single graph relaxed plan is reasonable when plans for achieving the goal from each world have high positive interaction. However, without high positive interaction the heuristic degrades quickly when the number of initial worlds increases.

With multiple planning graphs, *CAItAlt* is able to perform better in the Rovers domain, but takes quite a bit of time in the Logistics, BT, and BTC domains. In Rovers, capturing distance estimates for individual worlds and aggregating them by some means tends to be better than aggregating worlds and computing a single distance estimate (as in a single graph). In Logistics, part of the reason computing multiple graphs is so costly is that we are computing mutexes on each of the planning graphs. In BT and BTC, the total time increases quickly because the number of planning graphs, and relaxed plans for every search node increases so much as problems get larger.

Comparing the two multiple graph heuristics⁶ we demonstrate in *CAItAlt* namely h_{m-RP}^{MG} and h_{RPU}^{MG} , we notice the effect of our choices for state distance aggregation. The h_{m-RP}^{MG} relaxed plan heuristic aggregates state distances, as found on each planning graph, by taking the maximum distance, and the h_{RPU}^{MG} unions the relaxed plans, from each graph, and counts the number of actions in the unioned relaxed plan. As with the single graph relaxed plan, the h_{m-RP}^{MG} relaxed plan essentially measures one state to state distance; thus, performance suffers on the BT and BTC domains. However, using the unioned relaxed plan heuristic, we capture the independence among the multiple worlds so that we scale up better in BT and BTC. Despite the usefulness of the unioned relaxed plan, it is costly to compute and scalability is limited, so we turn to the *LUG* version of this same measure.

With the *LUG*, we use the $h_{RP}^{LUG(FX)}$ heuristic in *CAItAlt* heuristic. This heuristic uses a *LUG* with full cross world mutexes (denoted by *FX*). As in the similar h_{RPU}^{MG} heuristic, measuring overlap is important, and improving the speed of computing the heuristic tends to improve the scalability of *CAItAlt*. While *CAItAlt* is slower in the Rovers and BTC domains when using the *LUG*, we note that it is because of the added cost of computing cross world mutexes – we are able to improve the speed by relaxing the mutexes, as we will describe shortly.

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: computing no mutexes (NX), computing only static interference mutexes (StX), computing (StX) plus inconsistent support and competing needs mutexes – dynamic mutexes (DyX), and computing (DyX) plus induced mutexes – full mutexes (FX). The cross world checking (see appendix B) reduction schemes are:

6. We show h_{s-RP}^{MG} with *POND*.

Problem	h_0	h_{card}	h_{RP}^{SG}	h_{m-RP}^{MG}	h_{RP}^{MG}	$h_{RP}^{LUG(FX)}$
Rovers 1	2255/5	18687/14	543/5	542/5	185/5	15164/5
2	49426/8	TO	78419/8	8327/8	29285/9	32969/8
3	TO	-	91672/10	20162/10	2244/11	16668/10
4	-	-	TO	61521/16	3285/15	31584/13
5	-	-	-	TO	TO	TO
6	-	-	-	-	-	-
Logistics 1	1108/9	4268/9	198/9	183/9	1109/9	1340/9
2	TO	TO	7722/15	15491/15	69818/19	18535/15
3	-	-	3324/14	70882/14	TO	16458/15
4	-	-	141094/19	TO	-	178068/19
5	-	-	TO	-	-	TO
BT 2	19/2	14/2	18/2	20/2	21/2	12/2
10	4837/10	56/10	5158/10	8988/10	342/10	71/10
20	TO	418/20	TO	TO	2299/20	569/20
30	-	1698/30	-	-	9116/30	2517/30
40	-	5271/40	-	-	44741/40	7734/40
50	-	12859/50	-	-	TO	18389/50
60	-	26131/60	-	-	-	37820/60
70	-	48081/70	-	-	-	70538/70
80	-	82250/80	-	-	-	188603/80
BTC 2	30/3	16/3	16/3	33/3	23/3	18/3
10	15021/19	161/19	15679/19	41805/19	614/19	1470/19
20	TO	1052/39	TO	TO	2652/39	51969/39
30	-	3823/59	-	-	9352/59	484878/59
40	-	11285/79	-	-	51859/79	TO
50	-	26514/99	-	-	TO	-
60	-	55687/119	-	-	-	-
70	-	125594/140	-	-	-	-

Figure 11: Results for *CAItAlt* for conformant Rovers, Logistics, BT, and BTC. The data is Total Time / # Expanded Nodes, “TO” indicates a time out, “OOM” indicates out of memory, and “-” indicates no attempt.

computing mutexes across same-worlds (SX), computing mutexes across pairs of worlds in the intersection (conjunction) of element labels (IX).

Figure 12 shows that within *CAItAlt*, 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. To see if we could reduce graph construction overhead without hindering performance, we evaluated h_{RP}^{LUG} 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 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*.

POND: We show the total time and the number of expanded nodes for *POND* solving the conformant problems (including Ring and Cube Center) in Figure 13, and for *POND* solving the conditional problems in Figure 14. As with *CAItAlt* we show the total time and number of expanded nodes for each test. We also add the h_{s-RP}^{MG} heuristic, not implemented in *CAItAlt*, that sums the values of relaxed plans extracted from multiple planning graphs. We do not compute mutexes on any of the planning graphs used for heuristics in *POND* mainly because we build planning graphs for each search node. We proceed by first commenting on the performance of *POND*, with the different heuristics, in the conformant domains, then discuss the conditional domains.

In the conformant domains, *POND* generally does better than *CAItAlt*. This may be attributed in part to implementation-level details. Aside from a few differences that we will mention, we see similar trends in the performance of the various heuristics in both *CAItAlt* and *POND*. Namely, the *NG* and *SG* heuristics have limited ability to help the planner scale, the *MG* heuristics help the planner scale better but are costly, and the *LUG* provides the best scalability. The difference between the *MG* and the *LUG* are especially pronounced in Cube Center and Ring, where the size of the initial belief state is quite large as the instances scale. Interestingly in Ring, breadth first search and the single graph relaxed plan are able to scale due to reduced heuristic computation time and focussed search effort. The *LUG* is able to provide good search guidance, but tends to take a long time computing heuristics.

We are also now able to compare the choices for aggregating the distance measures from relaxed plans for multiple graphs. We see that taking the max of the relaxed plans, h_{m-RP}^{MG} , in assuming positive interaction among worlds is useful in Logistics and Rovers, but loses the independence of worlds in the BT and BTC domains. However, taking the sum of the relaxed plan values for different worlds, h_{s-RP}^{MG} is able to capture the independence in the BT domain. We notice that the summation does not help *POND* in the BTC domain; this is because we overestimate the heuristic value for some nodes by counting the Flush action once for each world when it in fact only needs to be done once (i.e. we miss

Problem	$h_{RP}^{LUG(NX)}$	$h_{RP}^{LUG(SIX)}$	$h_{RP}^{LUG(DgX)}$	$h_{RP}^{LUG(FX)}$	$h_{RP}^{LUG(DgX-SX)}$	$h_{RP}^{LUG(DgX-IX)}$	$h_{RP}^{LUG(FX-SX)}$	$h_{RP}^{LUG(FX-IX)}$
Rovers 1	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	TO	TO	32645/441/14	31293/291/14	14988/291/14	16772/291/14	14726/290/14	16907/290/14
5	-	-	698575/3569/45	TO	61373/3497/45	379230/3457/45	60985/3388/45	378869/3427/45
6	-	-	TO	-	217507/3544/37	565013/3504/37	225213/3408/37	588336/3512/37
Logistics 1	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	TO	TO	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	-	-	TO	TO	52036/2690/41	328114/4668/52	52109/2672/41	324508/4194/52
BT 2	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
BTC 2	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	TO	1211/6798/79	1824/6816/79	1253/6830/79	1068019/6940/79
50	260/15942/99	261/16452/99	TO	-	2890/16184/99	4412/16414/99	2926/16028/99	TO
60	435/32201/119	443/32923/119	-	-	6045/32348/119	9492/32350/119	6150/32876/119	-
70	742/62192/139	745/61827/139	-	-	TO	TO	TO	-

Figure 12: Results for *CALt* using h_{RP}^{LUG} with various mutex schemes for conformant Rovers, Logistics, BT, and BTC. The data is Graph Construction Time / All Non-Graph Construction Time / # Expanded Nodes, “TO” indicates a time out, “Om” indicates out of memory, and “-” indicates no attempt.

positive interaction). Finally, using the h_{RP}^{MG} heuristic we do well in every domain, aside from the cost of computing multiple graph heuristics, because we account for both positive

PLANNING GRAPH HEURISTICS FOR BELIEF SPACE SEARCH

Problem	h_0	h_{card}	h_{RP}^{SG}	h_{m-RP}^{MG}	h_{s-RP}^{MG}	h_{RP}^{MG}	h_{RP}^{LUG}
Rovers 1	540/36	520/21	590/6	580/6	580/6	580/6	590/6
2	940/249	790/157	700/15	1250/32	750/10	830/13	680/11
3	3340/1150	2340/755	3150/230	3430/77	1450/24	1370/23	850/16
4	TO	14830/4067	13480/1004	10630/181	7000/163	2170/34	1130/28
5	-	TO	TO	85370/452	12470/99	31480/73	2050/36
6	-	-	-	180890/416	15780/38	31950/73	9850/147
Logistics 1	560/169	530/102	680/46	970/58	730/21	650/9	560/9
2	TO	TO	TO	2520/32	6420/105	2310/20	910/15
3	-	-	-	27820/927	4050/83	2000/15	1130/14
4	-	-	-	5740/27	29180/211	53470/382	3180/46
5	-	-	-	42980/59	51380/152	471850/988	6010/42
BT 2	450/3	460/2	460/3	450/2	450/2	500/2	460/2
10	760/1023	590/428	1560/1023	6200/428	820/10	880/10	520/10
20	TO	TO	TO	TO	6740/20	6870/20	1230/20
30	-	-	-	-	41320/30	44260/30	4080/30
40	-	-	-	-	179930/40	183930/40	11680/40
50	-	-	-	-	726930/50	758140/50	28420/50
60	-	-	-	-	TO	TO	59420/60
70	-	-	-	-	-	-	113110/70
80	-	-	-	-	-	-	202550/80
BTC 2	460/5	460/4	450/5	460/4	460/3	470/3	460/3
10	1090/2045	970/1806	3160/2045	18250/1806	980/19	990/19	540/19
20	TO	TO	TO	TO	TO	9180/39	1460/39
30	-	-	-	-	-	54140/59	4830/59
40	-	-	-	-	-	251140/79	14250/79
50	-	-	-	-	-	1075250/99	34220/99
60	-	-	-	-	-	TO	71650/119
70	-	-	-	-	-	-	134880/139
CubeCenter 3	10/184	30/14	90/34	1050/61	370/9	0430/11	70/11
5	180/3198	20/58	3510/1342	60460/382	11060/55	14780/82	1780/205
7	1940/21703	40/203	46620/10316	TO	852630/359	1183220/444	27900/1774
9	TO	70/363	333330/46881	-	TO	TO	177790/7226
11	-	230/1010	TO	-	-	-	609540/17027
13	-	700/2594	-	-	-	-	TO
Ring 2	20/15	20/7	30/15	80/8	80/7	80/8	30/8
3	20/59	20/11	70/59	1500/41	500/8	920/19	70/10
4	30/232	20/15	350/232	51310/77	6370/11	19300/40	0250/24
5	160/973	20/19	2270/973	TO	283780/16	TO	970/44
6	880/4057	30/23	14250/4057	-	TO	-	4080/98
7	5940/16299	40/27	83360/16299	-	-	-	75020/574
8	39120/64657	40/31	510850/64657	-	-	-	388300/902
9	251370/261394	50/35	TO	-	-	-	TO
10	TO	70/39	-	-	-	-	-

Figure 13: Results for *POND* for conformant Rovers, Logistics, BT, and BTC. The data is Total Time / # Expanded Nodes, “TO” indicates a time out, “OoM” indicates out of memory, and “-” indicates no attempt.

interaction and independence by taking the overlap of relaxed plans. Again, with the *LUG*

Problem	h_0	h_{card}	h_{RP}^{SG}	h_{m-RP}^{MG}	h_{s-RP}^{MG}	h_{RPU}^{MG}	h_{RP}^{LUG}
Rovers 1	550/36	480/21	580/6	570/6	570/6	580/6	580/6
2	1030/262	550/36	780/15	760/14	710/12	730/12	730/13
3	1700/467	590/48	3930/248	830/15	830/15	910/17	810/16
4	5230/1321	620/58	6760/387	1020/20	1040/21	1070/21	910/21
5	TO	TO	TO	16360/175	11100/232	12810/209	7100/174
6	-	-	-	31870/173	24840/159	30250/198	13560/174
Logistics 1	530/118	TO	740/46	580/10	570/10	600/10	570/10
2	TO	-	TO	1630/30	1300/36	1360/36	1250/36
3	-	-	-	1360/20	1250/19	1290/19	1210/19
4	-	-	-	4230/59	3820/57	3940/57	4160/57
5	-	-	-	27370/183	19620/178	20040/178	20170/178
BT 2	460/5	460/3	450/3	460/3	450/3	470/3	460/3
10	TO	470/19	111260/7197	970/19	970/19	1020/19	550/19
20	-	510/39	TO	9070/39	9060/39	9380/39	1610/39
30	-	620/59	-	52410/59	52210/59	55750/59	5970/59
40	-	850/79	-	207890/79	206830/79	233720/79	17620/79
50	-	1310/99	-	726490/99	719000/99	TO	43020/99
60	-	2240/119	-	TO	TO	-	91990/119
70	-	24230/139	-	-	-	-	170510/139
80	-	45270/159	-	-	-	-	309940/159
BTC 2	450/6	460/3	470/5	470/3	460/3	470/3	470/3
10	TO	480/19	271410/10842	1150/19	1140/19	1200/19	590/19
20	-	510/39	TO	11520/39	TO	11610/39	1960/39
30	-	660/59	-	62060/59	-	64290/59	6910/59
40	-	970/79	-	251850/79	-	274610/79	19830/79
50	-	1860/99	-	941220/99	-	TO	49080/99
60	-	4010/119	-	TO	-	-	103480/119
70	-	7580/139	-	-	-	-	202040/139

Figure 14: Results for *POND* conditional Rovers, Logistics, BTS, BTCS. The data is Total Time / # Expanded Nodes, “TO” indicates a time out, “OoM” indicates out of memory, and “-” indicates no attempt.

relaxed plan, analogous to the multiple graph unioned relaxed plan, *POND* scales well because we measure overlap and lower the cost of computing the heuristic significantly.

The main change we see in using *POND* versus *CAItAlt* is that the direction of search is different, so the h_{card} heuristic performs unlike before. In the BT and BTC domains cardinality does not work well in progression because the size of belief states does not change as we get closer to the goal (it is impossible to ever know which package contains the bomb). However, in regression we start with a belief containing all states consistent with the goal and regressing actions limits our belief to only those states that can reach the goal through those actions. Thus in regression the size of belief states decreases, but in progression remain constant.

The performance of *POND* in the conditional domains exhibits similar trends to the conformant domains, with a few exceptions. Like the conformant domains, the *MG* relaxed

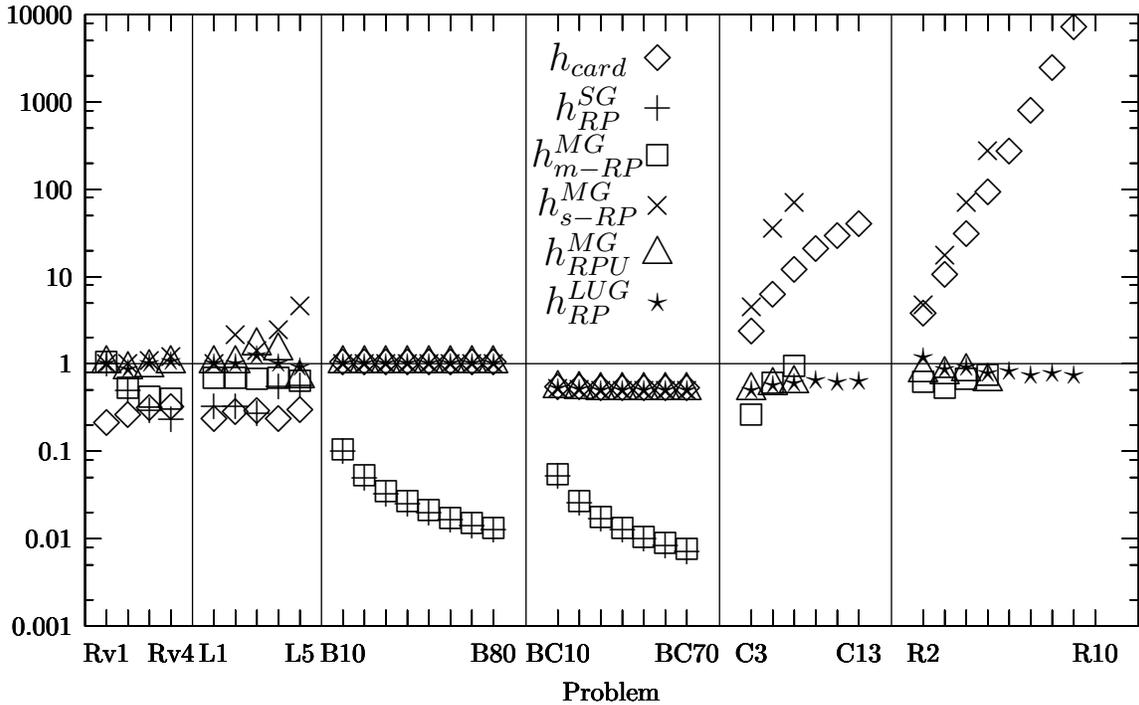


Figure 15: Ratio of heuristic estimates to optimal plan length. Rv = Rovers, L = Logistics, B = BT, BC = BTC, C = Cube Center, R = Ring.

plans tend to outperform the SG relaxed plan, but the LUG relaxed plan does best overall. Unlike the conformant domains, The h_{m-RP}^{MG} performs much better in BTS and BTCS over BT and BTC partly because the conditional plans have a lower average cost. The h_{card} heuristic does better in BTS and BTCS over BT and BTC because the belief states actually decrease in size when they are partitioned by sensory actions.

Finally, we compare the heuristic estimates for the distance between the initial belief state and the goal belief state for all the heuristics used in conformant problems solved by *POND*. Figure 15 shows the ratio of the heuristic estimate for ($h(BS_I)$) to the optimal serial plan length (i.e. $h^*(BS_I)$) in several problems. The points below the line (where the ratio is one) are under-estimates, and those above are over-estimates. Some of the problem instances are not shown because no optimal plan length is known.

We note that in all the domains the h_{RP}^{LUG} and h_{RPU}^{MG} heuristics are very close to h^* . Interestingly, h_{s-RP}^{MG} and h_{m-RP}^{MG} are both close to h^* in Rovers and Logistics; whereas the former is close in the BT and BTC problems, and the latter is close in CubeCenter and Ring. As expected, assuming independence (using sum) tends to over-estimate, and assuming positive interaction (using max) tends to under-estimate. The h_{RP}^{SG} heuristic tends

to under-estimate, and in some cases (CubeCenter and Ring) gives a value of zero (because there are initial states that satisfy the goal). The h_{card} heuristic is only accurate in BT and BTC, under-estimates in Rovers and Logistics, and over-estimates in Cube Center and Ring.

The accuracy of heuristics is in some cases disconnected from their run time performance. For instance h_{card} highly overestimates in Ring and Cube Center, but does well because the domains exhibit special structure and the heuristic is fast to compute. On the other hand, h_{RP}^{LUG} and h_{RPU}^{MG} are very accurate in many domains, but suffer in Ring and Cube Center because they can be costly to compute.

Intra-Planner Conclusions: Our findings fall into two main categories: one, what are effective estimates for belief state distances in terms of state to state distances, and two, how we can exploit planning graphs to support the computation of these distance measures.

In comparing ways to aggregate state distance measures to compute belief state distances, we found that measuring no interaction as in single graph heuristics tends to poorly guide planners, measuring independence and positive interaction of worlds works well in specific domains, and measuring overlap (i.e. a combination of positive interaction and independence) tends to work well in a large variety of instances.

Comparing graph structures that provide the basis for belief state distance measures, we found that the heuristics extracted from the single graph fail to systematically account for the independence or positive interaction among different possible worlds. Despite this lack in the distance measure, single graphs can still leverage some structure in domains like Rovers and Logistics. To more accurately reflect belief state distances, multiple graphs reason about reachability for each world independently. This accuracy comes at the cost of computing a lot of redundant MG structure and is limiting in instances with large belief states. Optimizing the representation of the redundant structure by using the LUG , planners are able to exhibit better scalability. The improvement in scalability is attributed to lowering the cost of heuristic computation, but retaining measures of multiple state distances. The LUG makes a trade-off of using an exponential time algorithm for evaluation of labels instead of building an exponential number of planning graphs. This trade-off is justified by our experiments.

5.2.2 CONFORMANT INTER-PLANNER COMPARISON

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 *CAItAlt* and *POND* to some of the other leading approaches to conformant planning. Note, since each approach uses a different planning representation (BDDs, GraphPlan, etc.), 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 input syntax; MBP, and KACMBP use

PLANNING GRAPH HEURISTICS FOR BELIEF SPACE SEARCH

Problem	CAItAlt $h_{RPU}^{LUG(DyX-SX)}$	POND h_{RP}^{LUG}	MBP	KACMBP	GPT	CGP	YKA	CFE
Rovers 1	16070/5	590/5	66/5	9293/5	3139/5	70/5	1220/7	70/5
2	10457/8	680/9	141/8	9289/15	4365/8	180/8	2050/10	30/8
3	10828/10	850/11	484/10	9293/16	5842/10	460/10	1740/12	10/10
4	15279/13	1130/16	3252/13	9371/18	7393/13	1860/13	2010/16	10/13
5	64870/29	2050/25	OoM	39773/40	399525/20	OoM	7490/27	18/22
6	221051/25	8370/25	727/32	TO	TO	-	24370/26	21/23
Logistics 1	907/9	560/9	37/9	127/12	916/9	60/6	250/13	10/9
2	2862/15	910/15	486/24	451/19	1297/15	290/6	670/19	12/15
3	10810/15	1130/14	408/14	1578/18	1711/11	400/8	20280/21	14/12
4	24862/19	3180/22	2881/27	8865/22	9828/18	1170/8	17530/27	12/18
5	54726/34	6010/29	OoM	226986/42	543865/28	TO	141910/40	25/28
BT 2	16/2	460/2	6/2	10/2	487/2	20/1	0/2	0/2
10	71/10	520/10	119/10	16/10	627/10	520/1	0/10	30/10
20	552/20	1230/20	80/20	84/20	472174/20	3200/1	20/20	4400/20
30	2415/30	4080/30	170/30	244/30	TO	10330/1	80/30	4500/30
40	7543/40	11680/40	160/40	533/40	-	24630/1	160/40	26120/40
50	17573/50	28420/50	300/50	1090/50	-	49329/1	250/50	84730/50
60	35983/60	59420/60	480/60	2123/60	-	87970/1	420/60	233410/60
70	67690/70	113110/70	730/70	3529/70	-	145270/1	620/70	522120/70
80	157655/80	202550/80	1080/80	1090/80	-	TO	3310/80	979400/80
BTC 2	16/3	460/3	8/3	18/3	465/3	0/3	10/3	10/3
10	89/19	540/19	504/19	45/19	715/19	39370/19	30/19	57/19
20	651/39	1460/39	98/39	211/39	-	-	240/39	2039/39
30	2721/59	4820/59	268/59	635/59	-	-	1210/59	23629/59
40	8009/79	14250/79	615/79	1498/79	-	-	3410/79	116156/79
50	19074/99	34220/99	1287/99	10821/99	-	-	8060/50	334879/99
60	38393/119	71650/119	2223/119	5506/119	-	-	15370/119	TO
70	65448/139	134880/139	3625/139	2640/139	-	-	27400/139	-
CubeCenter 3	TO	70/9	10/9	20/9	40/9	28990/3	0/9	20/15
5	-	1780/18	16/18	20/18	363/18	TO	0/19	28540/45
7	-	27900/29	35/27	70/27	4782/27	-	20/34	TO
9	-	177790/36	64/36	120/36	42258/36	-	80/69	-
11	-	609540/47	130/45	230/45	26549/45	-	190/68	-
Ring 2	TO	30/6	0/5	0/5	31/5	TO	0/5	360/12
3	-	70/8	0/8	40/8	35/8	-	0/8	TO
4	-	250/13	10/11	30/11	60/11	-	20/11	-
5	-	970/17	20/14	50/14	635/14	-	80/14	-
6	-	4080/22	30/17	120/18	51678/17	-	110/17	-
7	-	75020/30	80/20	230/21	TO	-	300/20	-
8	-	388300/29	160/23	600/24	-	-	480/23	-

Figure 16: Results for CAItAlt using $h_{RPU}^{LUG(DyX-SX)}$, POND using h_{RP}^{LUG} , MBP, KACMBP, GPT, CGP, YKA, and CFF for conformant Rovers, Logistics, BT, BTC, Cube Center, and Ring. The data is Total Time / # Plan Steps, “TO” indicates a time out, “OoM” indicates out of memory, and “-” indicates no attempt.

AR encodings which may give them an advantage in reducing the number of literals and actions. We gave the MBP planners the same grounded and filtered action descriptions

that we used in *CAItAlt* 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. Nevertheless, Figure 16 compares MBP, KACMBP, GPT, CGP, YKA, and CFF with $h_{RP}^{LUG(DyX-SX)}$ in *CAItAlt* and h_{RP}^{LUG} in *POND* with respect to run time and plan length.

MBP: The MBP planner uses a cardinality heuristic that in many cases overestimates plan distances (as per our implementation). It is interesting to note that in the more difficult problem instances in the Rovers and Logistics domains MBP and KACMBP tend to generate much longer plans than the other planners. MBP does outperform *POND* in some cases but does not find solutions in certain instances (like Rovers 5), most likely because of its heuristic. We note that KACMBP and MBP are quite fast on the Cube Center and Ring domains, but have more trouble on domains like Rovers and Logistics. This illustrates how a heuristic modelling knowledge opposed to reachability can do well in domains where the challenge is uncertainty not reachability.

Optimal Planners: The optimal approaches (CGP and GPT) tend not to scale as well, despite their good solutions. 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 parallel conformant plan depth increases.

YKA: YKA, like *CAItAlt* is a regression planner, but the search engine is very different and YKA does not compute heuristics. YKA proves to perform well on all the domains without heuristics because of its search engine. The relative performance between YKA and *POND* is similar to the relative performance of *CAItAlt* and *POND*, ascribing to the differences between progression and regression. Additionally, it seems YKA has a stronger regression search engine than *CAItAlt*. *POND* is able to do better than YKA in the Rovers and Logistics domains, but it is unclear that it is because of the search direction or heuristics.

CFF: Conformant FF, using a relaxed plan similar to the *LUG* relaxed plan, does very well in the Rovers and Logistics domains because it uses the highly optimized FF search engine as well as a cheap to compute relaxed plan heuristic. However, CFF does not do as well in the BT, BTC, Cube Center, and Ring problems because there are not as many literals that will be entailed by a belief state. A possible reason CFF suffers is our encodings. The Cube Center and Ring domains are naturally expressed with multi-valued state features, and in our transformation to binary state features we describe the values that must hold but also the values that must not hold. This is difficult for CFF because the conditional effect antecedents contain several literals and their heuristic is restricted to considering only one such literal. It may be that CFF is choosing the wrong literal or simply not enough literals

PLANNING GRAPH HEURISTICS FOR BELIEF SPACE SEARCH

Problem	POND h_{RP}^{LUG}	MBP	GPT	SGP	YKA
Rovers 1	580/5	3312/11	3148/5	70/5	3210/5
2	730/8	4713/75	5334/7	760/7	6400/7
3	810/8	5500/119	7434/8	TO	7490/8
4	910/10	5674/146	11430/10	-	11210/10
5	7100/19	16301/76	TO	-	TO
6	13560/22	OoM	-	-	-
Logistics 1	570/7	41/16	1023/7	5490/6	1390/8
2	1250/12	22660/177	5348/12	TO	TO
3	1210/9	2120/45	2010/8	-	TO
4	4160/15	OoM	TO	-	-
5	20170/22	-	-	-	-
BT 2	460/2	0/2	510/2	0/1	0/2
10	550/10	240/10	155314/10	70/1	20/10
20	1610/20	OoM	OoM	950/1	60/20
30	5970/30	-	-	4470/1	200/30
40	17620/40	-	-	13420/1	400/40
50	43020/50	-	-	32160/1	810/50
60	91990/60	-	-	90407/1	1350/60
70	170510/70	-	-	120010/1	2210/70
80	309940/80	-	-	TO	3290/80
BTC 2	470/2	20/2	529/2	10/2	0/4
10	590/10	280/10	213277/10	TO	210/12
20	1960/20	OoM	TO	-	2540/22
30	6910/30	-	-	-	13880/32
40	19830/40	-	-	-	46160/42
50	49080/50	-	-	-	109620/52
60	103480/60	-	-	-	221460/62
70	202040/70	-	-	-	41374/72

Figure 17: Results for *POND* using h_{RP-ha}^{LUG} , MBP, GPT, SGP, and YKA for conditional Rovers, Logistics, BT, and BTC. The data is Total Time / # Max possible steps in a execution, “TO” indicates a time out, “OoM” indicates out of memory, and “-” indicates no attempt.

to get effective heuristics. However in BT and BTC where we used only one literal in effect antecedents CFF still performs poorly.

5.2.3 CONDITIONAL INTER-PLANNER COMPARISON

Figure 17 shows the results for testing the conditional versions of the domains on *POND*, MBP, GPT, SGP, and YKA.

MBP: The *POND* planner is very similar to MBP in that it uses progression search. However, *POND* uses an AO* search, whereas the MBP binary we used uses a depth first And-Or search. The depth first search used by MBP contributes to highly sub-optimal max length branches (as much as an order of magnitude longer than *POND*). For instance, the plans generated by 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. MBP tends to not scale as well as *POND* in all of the domains we tested. A possible reason for the performance of MBP is that the Logistics and Rovers domains have sensory actions with execution preconditions, which prevent branching early and finding deterministic plan segments for each branch. We experimented with MBP using sensory actions without execution precondition and it was able to scale relatively better, but plan quality was still lacking.

Optimal Planners: GPT and SGP generate better solutions but very slowly. GPT does better on the Rovers and Logistics problems because they exhibit some positive interaction in the plans, but SGP does well on BT because its planning graph search is well suited for shallow, yet broad (highly parallel) problems.

YKA: We see that YKA fares similar to GPT in Rovers and Logistics, but has trouble scaling for other reasons. We think that YKA may be having trouble in regression because of sensory actions since it was able to scale reasonably well in the conformant version of the domains. Despite this, YKA proves to do very well in the BT and BTC problems.

5.2.4 OVERALL CONCLUSIONS

In our internal comparisons of heuristics within *CAItAlt* and *POND*, as well as external comparisons with several state of the art conformant and conditional planners we have learned many interesting lessons about heuristics for planning in the uniform probability belief space.

- Distance based for belief space search help control conformant and conditional 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.
- 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 conditional planner, *POND*, to scale up in conditional domains, despite the fact that the heuristic computation does not model observation actions.

Our work to this point has centered on distance measures for belief space when uncertainty over the initial state is the sole reason for using belief states. Next, we discuss how these techniques can be extended to handle non-deterministic actions in making belief state distance measures.

6. Extension to Non-Deterministic Actions

While the scope of our presentation and evaluation is restricted to planning with initial state uncertainty and deterministic actions, some of the planning graph techniques can be extended to include non-deterministic actions of the type described by Rintanen [2003]. Non-deterministic actions have effects that are described in terms of a set of outcomes. For simplicity, we consider Rintanen’s conditionality normal form, where actions have a set of conditional effects (as before) and each consequent is a mutually-exclusive set of conjunctions (outcomes) – one outcome of the effect will result randomly. We outline the generalization of our single, multiple, and labelled planning graphs to reason with non-deterministic actions.

Single Planning Graphs: Single planning graphs, that are built from approximate belief states, do not lend themselves to a straight-forward extension. A single graph ignores uncertainty in a belief state by unioning its literals to form the initial planning graph layer. Continuing with the single graph assumptions about uncertainty, it makes sense to treat non-deterministic actions as deterministic. Similar to how we use approximated a belief state as a set of literals to form the initial literal layer, we can assume that a non-deterministic effect adds all literals appearing in the effect as if the action were deterministic (i.e. gives a set of literals). Single graph relaxed plan heuristics thus remain unchanged.

Multiple Planning Graphs: Multiple planning graphs are very much like Conformant GraphPlan [Smith and Weld, 1998]. We can generalize splitting the non-determinism in the current belief into multiple initial literal layers to splitting the outcomes of non-deterministic effects into multiple literal layers. The idea is to root a set of new planning graphs at each level, where each has an initial literal layer containing literals supported by an interpretation of the previous effect layer. By interpretations of the effect layer we mean every possible set of effect outcomes. A set of effect outcomes is possible if no two outcomes are outcomes of the same effect. Relaxed plan extraction still involves extracting

a relaxed plan from each planning graph. However, since each planning graph is split many times (in a tree-like structure) a relaxed plan is extracted from each “path of the tree”.

We note that this technique is not likely to scale because of the exponential growth in redundant planning graph structure over time. Further, in our experiments CGP has enough trouble with initial state uncertainty. We note that we should be able to do much better with the *LUG*.

Labelled Uncertainty Graph: With multiple graphs we are forced to capture non-determinism through splitting the planning graphs not only in the initial literal layer, but also each literal layer that follows at least one non-deterministic effect. We saw in the *LUG* that labels can capture the non-determinism that drove us to split the initial literal layer in multiple graphs. As such, these labels took on a syntactic form that describes subsets of the states in our source belief state. In order to generalize labels to capture non-determinism resulting from uncertain effects, we need to extend their syntactic form. Our objective is to have a label represent which sources of uncertainty (arising from the source belief or effects) causally support the labelled item. We also introduce a graph layer \mathcal{O}_k to represent outcomes and how they connect effects and literals.

At first thought, it would seem natural to describe the labels for outcomes in terms of their affected literals, but this can lead to trouble. The problem is that the literals in effect outcomes are describing states at a different time than the literals in the projected belief state. Further, an outcome that appears in two levels of the graph is describing a random event at different times. Using state literals to describe all labels will lead to confusion as to which random events (state uncertainty and effect outcomes at distinct steps) causally support a labelled item. The pathological example is when we have an effect whose set of outcomes matches one-to-one with the states in the source belief. In such a case, by using labels defined in terms of state literals we cannot distinguish which random event (the state uncertainty or the effect uncertainty) is described by the label.

We have two choices for describing effect outcomes in labels. In both choices we introduce a new set of label variables to describe how a literal layer is split. These new variables will be used to describe effect outcomes in labels and will not be confused with variables describing initial state uncertainty. In the first case, these variables will have a one-to-one matching with our original set of literals, but can be thought of as time-stamped literals. The number of variables we add to the label function is on the order of $2F$ per level (the number of fluent literals – assuming binary). The second option is to describe outcomes in labels with a new set fluents, where each interpretation over the fluents is matched to particular outcome. In this case, we add on the order of $\log |\mathcal{O}_k|$ variables, where \mathcal{O}_k is the k^{th} outcome layer. It would actually be lower if many of the outcomes were from deterministic effects because there is no need to describe them in labels. The former approach is likely to introduce fewer variables when there are a lot of non-deterministic

effects and they affect quite a few of the same literals, and the latter will introduce fewer variables when there are relatively few non-deterministic effects whose outcomes are fairly independent.

With the generalized labelling, we can still say that an item is reachable from the source belief state when its label is entailed by the source belief. This is because even though we are adding variables to labels, we are implicitly adding the variables to the source belief state. For example, say we add a variable v to describe two outcomes of an effect. One outcome is labelled v , the other $\neg v$. We can express the source belief BS_p that is projected by the *LUG* with the new variable as $BS_p \wedge (v \vee \neg v) = BS_p$. An item labelled as $BS_p \wedge v$ will not be entailed by the projected belief (i.e. is unreachable) because only one outcome causally supports it. If both outcomes support the item, then it will be reachable.

Understanding reachability, we can determine the level from which to extract a relaxed plan. The relaxed plan procedure does not change much in terms of its semantics other than having the extra graph layer for outcomes. We still have to ensure that literals are causally supported in all worlds they are labelled with in a relaxed plan, whether or not the worlds are from the initial state uncertainty or supporting non-deterministic effects.

7. 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 *CAltAlt* 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_{m-RP}^{MG} heuristic because they both estimate the cost of the farthest (max distance) state by

looking at a deterministic relaxation of the problem. In comparison to GPT, *CAltAlt* 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], and KACMBP [Bertoli and Cimatti, 2002]. Alternative to *CAltAlt* but similar to *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. MBP supports both progression and regression in the space of belief states, while KACMBP is a purely progression planner. 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 cardinality 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 *CAltAlt* 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.

YKA [Rintanen, 2003] is a regression conditional planner using BDDs but does not use heuristics. Recently Rintanen has also developed related reachability heuristics that consider distances for groups of states, which do not rely on planning graphs [Rintanen, 2004].

More recently, there has been closely related work on heuristics for constructing conformant plans within the CFF planner [Brafman and Hoffmann, 2004]. The planner represents belief states implicitly through a set of known facts, the action history (leading to the belief), and the initial belief state. CFF builds a planning graph forward from the set of known literals to the goal literals and backwards toward the initial belief. In the planning graph, conditional effects are restricted to single literals in their antecedent to enable 2-cnf reasoning. From this planning graph, CFF extracts a relaxed plan that represents supporting the goal belief from all states in the initial belief. The biggest differences between the *LUG* and the CFF technique are that the *LUG* reasons only forward from the source belief

state (assuming an explicit, albeit symbolic, belief state), and the *LUG* does not restrict reasoning to antecedents with a single literal. As a result, the *LUG* does not lose the causal support information and does not have to perform backward reasoning to the initial belief state.

Finally, *CAltAlt* 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]. *CAltAlt* 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.

8. Conclusion

With the intent of establishing a basis for belief state distance estimates, we have:

1. Discussed how heuristic measures can aggregate state distance measures to capture positive interaction, negative interaction, independence, and overlap.
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.

Our intent in this work was to provide a formal basis for measuring the distance between belief states in terms of underlying state distances. We investigated several ways to aggregate the state distances to reflect various assumptions about the interaction of state to state trajectories. The best of these measures turned out to measure both positive interaction and independence, what we call overlap. We saw that planners using this notion of overlap tend to do well across a large variety of domains and tend to have more accurate heuristics.

We’ve also 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. In comparison to other approaches, we've also been able to demonstrate the utility of structured reachability heuristics in controlling plan length and boosting scalability for both conformant and contingent planning.

We intend to investigate three additions to this work. The first, 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 conditional problems.

The second addition will be to consider heuristics for stochastic planning problems. We have preliminary work [Bryce and Kambhampati, 2005] describing how to propagate cost information on the *LUG* to support cost-sensitive conditional planning. Given our ability to propagate numeric information on the *LUG*, we are currently adapting these heuristics and *POND* to handle non uniform probabilities. The extension involves adding probabilities to labels by using ADDs instead of BDDs, and redefining propagation semantics. The propagation semantics replaces conjunctions with products, and disjunctions with summations. A label represents a probability distribution over possible worlds, the probability of reaching the vertex is a summation over the possible world probabilities, and the expected cost of a vertex is the sum of products between cost vector partitions and the label. Relaxed plans, which previously involved weighted set covers with a single objective (minimizing cost) become multi-objective by trading off cost and probability.

In addition to cost propagation we have also recently extended the *LUG* within the framework of state agnostic planning graphs [Cushing and Bryce, 2005]. The *LUG* seeks to avoid redundancy across the multiple planning graphs built for states in the same belief state. We extended this notion to avoid redundancy in planning graphs built for every belief state. We have shown that the state agnostic *LUG* (*SLUG*) which is built once per search episode (opposed to a *LUG* at each node) can reduce heuristic computation cost without sacrificing informedness.

Acknowledgments We would like to thank Minh B. Do, Romeo Sanchez, Terry Zimmermam, and Satish Kumar Thittamaranahalli, 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, the NSF grant IIS-0308139, as well as the 2003 NASA RIACS SSRP, and an IBM faculty award.

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.

- Piergiorgio 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.
- Piergiorgio 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 European 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.
- Ronen Brafman and Joerg Hoffmann. Conformant planning via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*, 2004.
- Daniel Bryce and Subbarao Kambhampati. Heuristic guidance measures for conformant planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, June 2004.
- Daniel Bryce and Subbarao Kambhampati. Cost sensitive reachability heuristics for handling state uncertainty. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI-05)*, 2005.
- Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In *Preedings of the 6th European Conference on Planning*, 2001.

- T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
- William Cushing and Daniel Bryce. State agnostic planning graphs. In *Proceedings of the Twenty-fifth National Conference on Artificial Intelligence (AAAI-05)*, 2005.
- 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.
- 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.
- D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *JAIR*, 20:1–59, 2003.
- 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.
- Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.
- 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)*, Acapulco, Mexico, August 2003.
- Jussi Rintanen. Distance estimates for planning in the discrete belief space. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 04)*, 2004.

David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (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. Additional Heuristics

For completeness, we present some additional heuristics adapted from classical planning to reason about belief state distances in each type of planning graph. Many of these heuristics appeared in our previous work [Bryce and Kambhampati, 2004]. We show how to compute the max, sum, and level heuristics on the single graph SG , multiple graphs MG , and the labelled uncertainty graph LUG . While these heuristics tend to be less effective than the relaxed plan heuristics, we provide them as reference.

A.1 Single Planning Graph Heuristics (SG)

Like, the relaxed plan for the single unmodified planning graph, we cannot aggregate state distances because all notion of separate states is lost in forming the initial literal layer.

No State Aggregation:

- **Max** In classical planning, the maximum cost literal is used to get a max heuristic, but we have formulae describing our beliefs, so we take the maximum cost clause as the cost of the belief to find the max heuristic h_{max}^{SG} . The maximum cost clause of the destination belief state, with respect to a single planning graph, is:

$$h_{max}^{SG}(BS_i) = \max_{C \in \kappa(BS_i)} cost(C)$$

where the cost of a clause is

$$cost(C) = \min_{l \in C} \min_{k: l \in \mathcal{L}_k} k$$

Here we find the cheapest literal as the cost of each clause to find the max cost clause. This is an underestimate of the closest state to our current belief.

- **Sum** Like the classical planning sum heuristic, we can take the sum h_{sum}^{SG} of the costs of the clauses in our belief state to estimate our belief state distance:

$$h_{sum}^{SG}(BS_i) = \sum_{C \in \kappa(BS_i)} cost(C)$$

This heuristic sums the costs of the literals of the closest estimated state in the belief state, and is inadmissible because there may be a single action that will support every clause, and we could count it once for each clause.

- **Level** When we have mutexes on the planning graph, we can compute a level heuristic h_{level}^{SG} (without mutexes the level heuristic is equivalent to max). The level heuristic maintains the admissibility of the max heuristic but improves the lower bound

by considering what level of the planning graph all literals in a constituent are non-pairwise mutex. The level heuristic is computed by taking the minimum among the $S \in \xi(BS_i)$, of the first level ($lev(S)$) in the planning graph where literals of S are present with none of them marked pairwise mutex. Formally:

$$h_{level}^{SG}(BS_i) = \min_{S \in \xi(BS_i)} lev(S)$$

Positive Interaction Aggregation:

- Individual state to state distances can not be computed on the single planning graph because all notion of which state in our current belief is used for the estimate is lost when the literals are unioned to form the initial layer.

Independence Aggregation:

- See Positive Interaction Aggregation.

Overlap Aggregation:

- See Positive Interaction Aggregation.

A.2 Multiple Planning Graph Heuristics (MG)

Similar to the various relaxed plan heuristics for the multiple graphs, we can compute a max, sum, or level heuristic on each of the multiple planning graphs and aggregate them with a max or sum to respectively measure positive interaction or independence. The reason we cannot aggregate the individual graph heuristics to measure overlap is that they are numbers, not sets of actions. Measuring overlap involves taking the union of heuristics from each graph and the union of numbers is not meaningful like the union of action sets from relaxed plans.

No Aggregation:

- There is no reason to use multiple graphs if there is no state distance aggregation. A single planning is sufficient.

Positive Interaction Aggregation:

- **Max** The max heuristic h_{m-max}^{MG} is computed with multiple planning graphs to measure positive interaction in the h_{m-max}^{MG} heuristic. This heuristic computes the max

cost clause in $\kappa(BS_i)$ for each graph $\gamma \in \Gamma$, similar to how $h_{m-max}^{SG}(BS_i)$ is computed, and takes the maximum. Formally:

$$h_{m-max}^{MG}(BS_i) = \max_{\gamma \in \Gamma} (h_{max}^{\gamma}(BS_i)) \quad (\text{A-1})$$

The h_{m-max}^{MG} heuristic considers the minimum cost, relevant literals of a belief state (those that are reachable given a possible world for each graph γ) to get state 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).

- **Sum** The sum heuristic that measures positive interaction for multiple planning graphs is h_{m-sum}^{MG} . $h_{m-sum}^{MG}(BS_i)$ computes the sum of the cost of the clauses in $\kappa(BS_i)$ for each graph $\gamma \in \Gamma$ and takes the maximum. Formally:

$$h_{m-sum}^{MG}(BS_i) = \max_{\gamma \in \Gamma} (h_{sum}^{\gamma}(BS_i))$$

h_{m-sum}^{MG} considers the minimum cost, relevant literals of a belief state (those that are reachable given the possible worlds represented for each graph γ) to get state measures. As with h_{m-max}^{MG} , the max is taken to estimate for the most costly world.

- **Level** Similar to h_{m-max}^{MG} and h_{m-sum}^{MG} , the $h_{m-level}^{MG}$ heuristic is found by first finding h_{level}^{γ} for each graph $\gamma \in \Gamma$ to get a state distance measure, and then taking the max across the graphs. $h_{level}^{\gamma}(BS_i)$ is computed by taking the minimum among the $S \in \xi(BS_i)$, of the first level $lev^{\gamma}(S)$ in the planning graph γ where literals of \hat{S} are present with none of them marked mutex. Formally:

$$h_{level}^{\gamma}(BS_i) = \min_{S \in \xi(BS_i)} lev^{\gamma}(S)$$

and

$$h_{m-level}^{MG}(BS_i) = \max_{\gamma \in \Gamma} (h_{level}^{\gamma}(BS_i))$$

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 $h_{m-level}^{MG}$, but is computed with dynamic programming rather than planning graphs.

Independence Aggregation:

- All heuristics mentioned for Positive Interaction Aggregation can be augmented to take the sum of costs found on the individual planning graphs rather than the max. We denote them as: h_{s-max}^{MG} , h_{s-sum}^{MG} , and $h_{s-level}^{MG}$. None of these heuristics are admissible because the same action may be used in all worlds, but we count its cost for every world by using summation.

Overlap Aggregation:

- See relaxed plans covered in Section 4.

A.3 Labelled Uncertainty Graph (*LUG*)

The max, sum, and level heuristics for the *LUG* are similar to the analogous multiple graph heuristics. The main difference with these heuristics for the *LUG* is that it is much easier to compute positive interaction measures than independence measures. The reason positive interaction is easier to compute is that we find the cost of a clause for all states in our belief at once, rather than on each of multiple planning graphs.

No Aggregation:

- The *LUG* is useful for aggregating the planning graphs for multiple states into one structure, so it does not make sense to throw away this information and compute a heuristic that does not aggregate states. Rather, if one has only one state under consideration, then a single graph would be ideal.

Positive Interaction Aggregation:

- **Max** The max heuristic h_{m-max}^{LUG} for the *LUG* finds the maximum clause cost across clauses of the current belief BS_i . The cost of a clause is the first level it becomes reachable. Formally:

$$h_{m-max}^{LUG}(BS_i) = \max_{C \in \kappa(BS_i)} \left(\min_{k: BS_P \models \ell_k^*(C)} k \right)$$

- **Sum** The sum heuristic h_{m-sum}^{LUG} for the *LUG* sums the individual levels where each clause in $\kappa(BS_i)$ is first reachable. Formally:

$$h_{m-sum}^{LUG}(BS_i) = \sum_{C \in \kappa(BS_i)} \left(\min_{k: BS_P \models \ell_k^*(C)} k \right)$$

- **Level** The level heuristic $h_{m-level}^{LUG}$ is the index of the first level where BS_i is reachable. Formally:

$$h_{m-level}^{LUG}(BS_i) = \min_{k:BS_P \models \ell_k^*(BS_i)} i$$

Independence Aggregation:

- All heuristics mentioned for Positive Interaction Aggregation can be augmented to take the sum of costs for each state in our belief. This may be inefficient due to the fact that we lose the benefit of having a *LUG* by evaluating a heuristic for each state of our current belief, rather than all states at once as in the Positive Interaction Aggregation. In such a case we are doing work similar to the multiple graph heuristic extraction, aside from the improved graph construction time. The Positive Interaction Aggregation is able to implicitly calculate the max over all worlds for most of the heuristics, whereas for sum we need to explicitly find a cost for each world. We denote the sum heuristics as: h_{s-max}^{LUG} , h_{s-sum}^{LUG} , and $h_{s-level}^{LUG}$.

State Overlap Aggregation:

- Covered in section 4.

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]. Cross world mutexes are useful to capture negative interactions in belief state distance measures (mentioned in Section 3). 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 $(\hat{\ell}_k(x) = S, \hat{\ell}_k(x') = S')$.

We can say that a formula f is reachable from our projected belief state BS_P , when considering cross-world mutexes, if for every pair of states in BS_P , f is reachable. For, a pair of states S and S' , f is reachable if i) $BS_P \models \ell_k^*(f)$, ii) there is a pair of states in f , say S'' and S''' such that $S \models \ell_k^*(S'')$ and $S' \models \ell_k^*(S''')$, iii) there are no two literals in either S'' or S''' same-world mutex in the respective worlds S and S' , and iv) there is not mutex between literals in S'' and S''' , in the respective worlds S and S' . There is a mutex between a pair literals l and l' , respectively from S'' and S''' if there is a mutex $(\hat{\ell}_k(l), \hat{\ell}_k(l'))$ such that $S \models \hat{\ell}_k(l)$ and $S' \models \hat{\ell}_k(l')$.

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{A}_k : The action mutexes can now hold for actions that are executable in different possible worlds.

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

$$\exists_{l \in \rho^e(a), l' \in \rho^e(a')} (\hat{\ell}_k(l) = S, \hat{\ell}_k(l') = S')$$

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

- **Interference** Effect interference mutexes do not change for cross-world mutexes, except that there is a pair of labels where $(\hat{\ell}_k(\varphi^i(a)) = BS_P, \hat{\ell}_k(\varphi^j(a')) = BS_P)$, instead of a single label.
- **Competing Needs** Effect competing needs mutexes change for cross-world mutexes because two effects $\varphi^i(a)$ and $\varphi^j(a')$, in worlds S and S' respectively, could be competing. Formally, a cross-world competing needs mutex $(\hat{\ell}_k(\varphi^i(a)) = S, \hat{\ell}_k(\varphi^j(a')) = S')$ exists between $\varphi^i(a)$ and $\varphi^j(a')$ in worlds S and S' if:

$$\exists_{l \in \rho^i(a), l' \in \rho^j(a')} (\hat{\ell}_k(l) = S, \hat{\ell}_k(l') = S')$$

- **Induced** 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 $(\hat{\ell}_k(\varphi^i(a)), \hat{\ell}_k(\varphi^h(a')))$, and $\varphi^i(a)$ induces $\varphi^j(a)$, then the mutex $(f \wedge \hat{\ell}_k(\varphi^i(a)), \hat{\ell}_k(\varphi^h(a')))$ holds.

An example of a cross-world induced effect mutex is shown in Figure 18. 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 $\varphi^i(a)$ induces

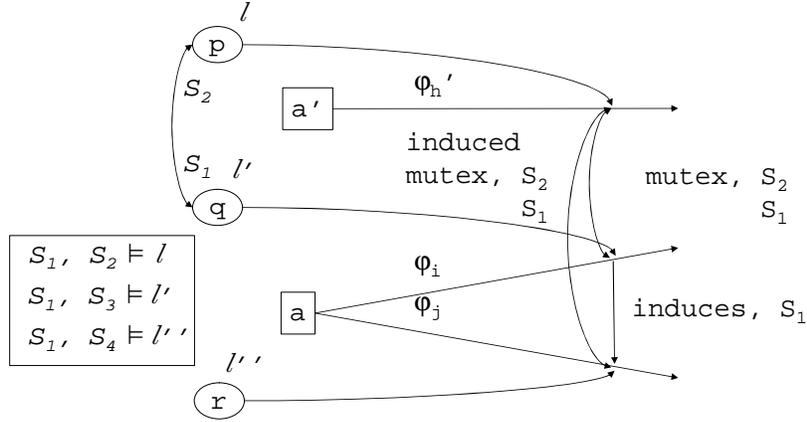


Figure 18: Example of a cross-world induced effect mutex.

$\varphi^j(a)$ in possible world S_1 , $\varphi^h(a')$ is mutex with effect $\varphi^i(a)$ across possible worlds (S_2, S_1) because of the mutex between p and q , and $\varphi^j(a)$ becomes induced mutex with $\varphi^h(a')$ 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:

$$\forall_{\substack{\varphi_i, \varphi^j(a') \in \mathcal{E}_{k-1}, \\ \text{where } l \in \varepsilon^i(a), l' \in \varepsilon^j(a')}} (\hat{\ell}_{k-1}(\varphi^i(a)) = S, \hat{\ell}_{k-1}(\varphi^j(a')) = S')$$