Belief Space Reachability Heuristics for Conformant and Contingent Planning

Daniel Bryce & Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University, Brickyard Suite 501
699 South Mill Avenue, Tempe, AZ 85281
{dan.bryce, rao}asu.edu

David E. Smith
NASA Ames Research Center
Comp. Sciences Division, MS 269-2
Moffett Field, CA 94035-1000
de2smith@email.arc.nasa.gov

Abstract

Although reachability heuristics have been shown to be useful in conformant an contingent planning, the cost of heuristic computation is still an issue. We present several improvements of our previous work on planning graph heuristics for planning in belief space. One, we generalize our approach, based on using multiple planning graphs, to symbolically represent many planning graphs in a structure we call a "labelled uncertainty graph" (LUG). The LUG allows us to compactly represent a set of planning graphs, needed to extract a conformant relaxed plan. Two, we further generalize the LUG to represent a larger set of planning graphs, one for each state in the reachable state space, in a structure called the "global labelled uncertainty graph" ($GLUG$). The $GLUG$ represents every planning graph we will ever need for a search episode and is only computed once. Three, we present a post-processing routine to include sensory actions into conformant relaxed plans, making them contingent relaxed plans. We have implemented and evaluated the LUG and relaxed plans in a planner called $POND$. We compare $POND$ to several state of the art planners and show our approach is quite competitive.

Introduction

Planning in non-deterministic and partially observable worlds is typically posed as search in belief space, where search nodes are sets of states (belief states). Given the effectiveness of reachability heuristics in controlling search in classical planning, in our recent work [Bryce and Kambhampati, 2004] we examined the issues involved in extending reachability heuristics to search in belief space. In particular, we developed a spectrum of planning graph heuristics for finding increasingly accurate estimates of belief-state distance. The most effective of these involves computing relaxed plans simultaneously over multiple planning graphs (corresponding to the individual states in a belief state). While this approach has lead to promising results, its scalability has been limited by the prohibitive cost of constructing multiple planning graphs. Another limitation was that our heuristics do not account for the presence of sensing actions, and are thus not as effective in guiding contingent planners. In this paper, we describe three key innovations that significantly reduce the cost of computing the reachability heuristics and extend their coverage to include contingent planning.

- To avoid building multiple planning graphs explicitly, we present a novel idea called "labelled uncertainty graph" (LUG), that symbolically represents the multiple planning graphs within a single planning graph. Loosely speaking, this single graph unions the causal support information present in the multiple graphs and pushes the disjunction, describing sets of possible worlds, into "labels" (ℓ). The graph is built efficiently by propagating labels, for sets of worlds, through each layer. The structure exhibits considerable space and time savings because the graph elements, which were previously constructed and represented (potentially many times) in multiple graphs, are constructed and represented only once. Labels indicate which of the explicit planning graphs a LUG element symbolically represents. We extend the relaxed plan extraction procedure so it operates directly on the LUG.
- We develop a further (natural) extension of LUG called "Global LUG" (or, more euphonically, $GLUG$), which in effect starts from a belief state corresponding to all states of the world. The LUG for any given belief state BS can be computed easily as a symbolic restriction of the $GLUG$ (as we shall show). This extension turns out to be particularly useful for progression search in belief space, where, without $GLUG$, we will have to compute a LUG for each belief state encountered during search. With the $GLUG$ we can attain significant time savings because we avoid repeated construction of graph components that are common to LUGs built for different search nodes.
- To provide more accurate estimates in the presence of sensing actions, we develop the notion of "contingent relaxed plans" which are extracted from the conformant ones that we build from the LUG. Our extension is motivated by an attempt to capture two aspects of contingent plans that are not measured by conformant relaxed plans. The first is that we do not have to execute all actions in a conformant relaxed plan if we can sense and decide which actions are irrelevant, and the second is that sensing may itself require costly set-up actions. The first concern is that we are overestimating the heuristic for a search node with a conformant relaxed plan because we may be able to use a sensory action later in the search and the actual plan suffix has a lower expected cost. The second concern is that a conformant relaxed plan underestimates cost when sensing is not free because ignoring

sensors also means ignoring the cost of the sensor's causal support. We present initial work on a conformant relaxed plan post-processing procedure to capture these notions in a contingent relaxed plan.

We have implemented all these heuristics and have tested them on a progression planner that we call $POND$–which can generate both conformant and contingent plans. Our experiments with $POND$ show that heuristics based on the LUG and $GLUG$ lead to significant scale-ups over those based on multiple planning graphs ones, and that contingent relaxed plans can help improve plan quality as well as scalability in problems with difficult to support sensory actions.[1] We also report on comparison studies with several state of the art conformant and contingent planners. These results show that $POND$ with our heuristics is competitive with the best conformant planners, and is able to produce significantly higher quality contingent plans. Our results are particularly encouraging considering that our current implementation of $POND$ does not take advantage of the most efficient (BDD-based) child generation techniques, as in [Bertoli *et al.*, 2001a].

We proceed by describing the representation we use in $POND$ and the search algorithm. The main contribution of the paper, regarding the LUG construction, conformant relaxed plan extraction, and sensory action insertion follows. We then evaluate $POND$ with respect to several planners, on several existing planning domains, as well as new domains, to show that our approach is scalable while retaining plan quality. We conclude with discussion of related work and directions for further research.

Representation

Our planning formulation in $POND$ uses progression search to find strong conformant and contingent plans, with the assumption of partial observability. A strong plan guarantees that after a finite number of possibly nondeterministic 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. Contingent plans are a more general case where the plan is structured as a graph. Contingent plans have graph structure because they include sensory actions. In this presentation, we restrict contingent plans to DAGs, but there is no conceptual reason why they cannot be general graphs.

We formulate $POND$'s search in the space of belief states, a technique first 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. A solution plan's quality is measured by its expected execution length, assuming each branch is equally likely.

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 also consider two normal forms of a logical formula f, the first is $\kappa(f)$ which is the CNF, and the second is $\hat{\xi}(f)$ which is the a DNF. The CNF form is seen as a conjunction of "clauses" C each of which are a disjunction of literals, while the DNF form is seen as a disjunction of "constituents" $\hat{S}$ each of which is a conjunction of literals.[2] We find it useful to think of CNF and DNF represented as sets – CNF is a conjunctive set of clauses while DNF is a disjunctive set of constituents.

A very important point to understand is that we use canonical forms of CNF and DNF when referring to $\kappa(.)$ and $\hat{\xi}(.)$. We use a BDD package [Brace *et al.*, 1990] to obtain these canonical forms. The package allows us represent a general propositional formula as a ROBDD, from which we can obtain the canonical forms as a set of minterms (DNF) and maxterms (CNF)[Meinel and Theobald, 1998].

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 fluent literals, and is also referred to as a set of possible worlds. A state S is in the set of states represented by a belief state BS if S is a model of 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.

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

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

[1] We have also experimented with these heuristics in the C*AltAlt* regression planner where we also got interesting results.

[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.

actions a are described in terms of an executability precondition ρ_e and a set of effects, Φ_a. The executability precondition, ρ_e, is a propositional formula that must hold for the action to be executable.

Our causative action representation is similar to the action formalism described by Rintanen [2003], using the conditionality normal form. Causative actions have a set of conditional effects where each conditional effect φ_i is of the form $\rho_i \implies (\varepsilon_{i,0}|...|\varepsilon_{i,j})$, where the antecedent ρ_i is a general formula and a consequent outcome $\varepsilon_{i,j}$ is a conjunction of literals. When $j > 0$ the effect is non-deterministic. By convention the unconditional effects are described as $\rho_0 = \top$ and a given $(\varepsilon_{0,0}|...|\varepsilon_{0,j})$.[4]

Sensory actions have a set $\Phi_a = \{o_0, ..., o_i\}$ of observational effect formulas. Each observational effect formula, o_i, defines the properties of an outcome of the sensor.

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

$DunkP1 : \langle \rho_e = \neg clog, \Phi_{DunkP1}\{clog, inP1 \implies \neg arm\}\rangle$
$DunkP2 : \langle \rho_e = \neg clog, \Phi_{DunkP2} = \{clog, inP2 \implies \neg arm\}\rangle$
$Flush : \langle \rho_e = \top, \Phi_{Flush} = \{\neg clog\}\rangle$
$DetectMetal : \langle \rho_e : \top, \Phi_{DetectMetal} = \{inP1, \neg inP1\}\rangle$

Progression: We progress actions over belief states to generate a set of successor belief states. An action a progressed 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, $BS \not\models \rho_e$, 2) a single belief state if a is a causative action (even when the action is non-deterministic), or 3) several belief states if a is a sensory action.

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 state $S \in \mathcal{M}(BS)$. If an action has non-deterministic effects, we generate a result state for every non-deterministic outcome. The case when an action is sensory is efficient for progression because we obtain a set of successors by individually taking the conjunction of each observational effect o_i with BS.

To illustrate progression, consider the successor sets $B_1 = \{BS_{11}\}$ for $Progress(BS_I, DunkP1)$ and $B_2 = \{BS_{21}, BS_{22}\}$ for $Progress(BS_I, DetectMetal)$. We first progress $DunkP1$, which is applicable to BS_I because $BS_I \models \neg clog$. The first state of BS_I is progressed by $DunkP1$ to get the result: $(clog \wedge \neg arm \wedge inP1 \wedge \neg inP2)$. The second state of BS_I, is progressed by $DunkP1$ to get a result: $(clog \wedge arm \wedge \neg inP1 \wedge inP2)$. The two results are taken together to get: $BS_{11} = (clog \wedge \neg arm \wedge inP1 \wedge \neg inP2) \vee (clog \wedge arm \wedge \neg inP1 \wedge inP2)$. We next progress $DetectMetal$ by taking the conjunction of its observation formula with BS_I. For $o_0 : inP1$ we get $BS_{21} = arm \wedge \neg clog \wedge inP1 \wedge \neg inP2$ and for $o_1 : \neg inP1$ we get $BS_{22} = arm \wedge \neg clog \wedge \neg inP1 \wedge inP2$.

For efficiency reasons however, we would like to implement progression directly in terms of the logical formula corresponding to BS. In $POND$, we currently do this in terms of the first-principles progression over every state in a belief state. We should note however that it is often even more efficient to do the progression in terms of the BDD representation of the belief state (as done in MBP [Cimatti and Roveri, 2000]). We are currently working on making $POND$ do progression on BDDs and once this is done the relative performance of $POND$ with respect to other planners will improve even more.

[4]For lack of space we do not discuss or evaluate our heuristics for non-deterministic actions. Please refer to an extended version of this paper at: http://rakaposhi.eas.asu.edu/belief-search/.

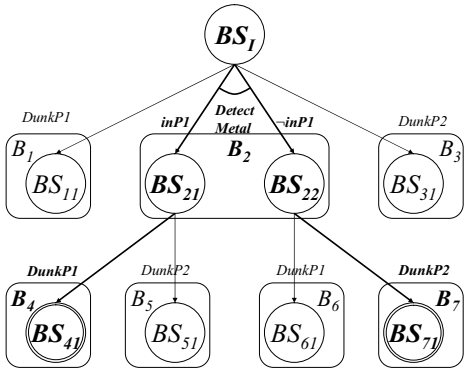

Figure 1: The AO* search graph for $BTCS$. The plan includes the bolded components.

POND Search

We use progression-based AO* search [Nilsson, 1980], in the $POND$ planner to generate conformant and contingent plans. In the search graph, the nodes are belief states and the AND-connectors (also commonly referred to as hyperedges or k-connectors) are actions. We need AO* because the application of a sensing action to a belief state in essence partitions the belief state. We use AND-connectors for actions because sensory actions have several outcomes, all if any of which must be included in a solution.

The AO* algorithm starts with the initial belief state as the only node in the search graph. The algorithm consists of a main outer loop that expands a node in the search graph, then updates the marked "best" edges and solved nodes of the graph. The algorithm terminates when the updating procedure marks the initial belief state as solved. A node is solved when all of its marked conjunctive descendants are solved, or it is a goal node. The final set of marked edges corresponds to our plan.

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 expectation rather than a sum, and use a weighted f-value. The first difference is to ensure that plans are strong, the second is to guide search toward plans with lower expected execution length, rather than a best worst-case execution length, and the third is to bias our search to trust the heuristic function. Our motivation for finding plans with lower expected length is that we are likely to expand a shallower search graph, and it doesn't change our semantics to assume a uniform probability distribution over sensory outcomes when there is no distribution provided.

As an example of search in $POND$, consider the $BTCS$ example whose search graph is shown in Figure 1. The

search graph initially contains only BS_I to which we assign a cost, $f(BS_I) = 2$, via a heuristic function, h, which we will describe in the next section. The successor sets of BS_I are $\{B_1, B_2, B_3\}$, for actions $DunkP1, DetectMetal$, and $DunkP2$, respectively – B_3 contains a belief state BS_{31}. We find a cost for each new node with our heuristic function, giving: $f(BS_{11}) = 2, f(BS_{21}) = 1, f(BS_{22}) = 1, f(BS_{31}) = 2$. We then update the cost for BS_I to be $1 + min(2, (1 + 1)/2, 2)$. Since B_2 contributed to minimizing the cost of BS_I, $DetectMetal$ is marked as the best action. Next, we determine the next node to expand, which is BS_{21}. The successor sets of BS_{21} are $\{B_4, B_5\}$, each with a single element for actions $DunkP1$, and $DunkP2$, respectively. We find a cost for each, giving: $f(BS_{41}) = 0, f(BS_{51}) = 2$. We also find that BS_{41} is labelled solved because $BS_{41} \models BS_G$. We then update the cost of BS_{21} to be the $1 + min(0, 2)$. Since B_4 contributed to minimizing the cost of BS_{21}, the $DunkP1$ action is marked. We backup costs and solved labels to BS_I, but BS_I is not marked solved because its best action's other successor BS_{22} is not marked solved. We find the last unexpanded node, BS_{22} and expand it to get $\{B_6, B_7\}$, each with a single element, for actions $DunkP1$, and $DunkP2$, respectively. We find a cost for each new node, giving: $f(BS_{61}) = 2, f(BS_{71}) = 0$. We find that BS_{71} is labelled solved because $BS_{71} \models BS_G$. We backup costs and solved labels to BS_I, and BS_I is now marked solved because BS_{21} and BS_{22} are both marked solved.

Our plan is: $DetectMetal$,if$(inP1)\{DunkP1\}$, else if$(\neg inP1)\{DunkP2\}$.

Labelled Uncertainty Graph (LUG)

The $POND$ search algorithm uses a heuristic function to assign initial costs to search nodes. Naturally, the heuristic function is a very important consideration for search and it needs to reflect the reachability measures for features of our search space. Since we are performing progression search over sets of states, to reach a set of goal states, the heuristic should reason about the distance between a source set of states and a destination set of states, a so called belief state to belief state distance [Bryce and Kambhampati, 2004].

Previously [Bryce and Kambhampati, 2004], we built a planning graph for each constituent that was part of our source belief state's constituent representation. We built each graph until it reached a state in the destination belief state, then we extracted a relaxed plan. We finished by aggregating the relaxed plans to get a distance measure. We now generalize this approach to symbolically represent the multiple graphs in one labelled graph and extract a relaxed plan that has the same properties as our aggregated relaxed plan.

Before discussing the labelled graph we present some notation to aid a generalization we make later. Since we present two ways of using the LUG, one where the LUG is built for each search node, and another where the LUG is built once for entire search, we define construction in terms of a general belief state BS_P, denoting the set of states from which we project. We also refer to a belief state BS_i, denoting the belief state for which we are obtaining a heuristic measure. When building the LUG for each search node, $BS_i = BS_P$, but when building the $GLUG$ we only require $BS_i \models BS_P$. Finally, the label of a graph element in level k is denoted $\ell_k(.)$.

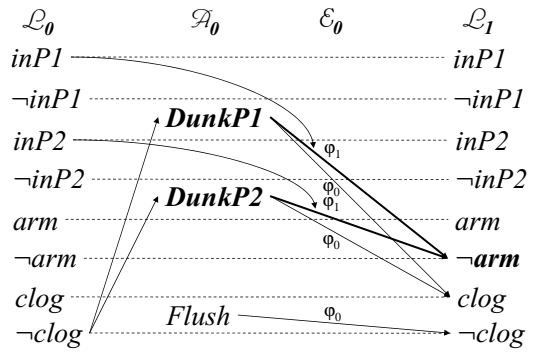

Figure 2: A portion of the LUG for $BTCS$ without labels. The first level is not shown because it is similar to the zeroth. The labels of individual elements are described below. A relaxed plan to support $\neg arm$ from BS_I includes the bolded components

The LUG and $GLUG$ have an initial layer consisting of every literal for every fluent in F. In the initial layer the label $\ell_0(l)$ of each literal l is identical to $l \wedge BS_P$, which represents 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 of labels though layers until no label of any literal changes between layers, a condition referred to as level off.

An important notion for our discussion of labels is when a literal or formula is *reachable*.[5] We can 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 $BS_i \models \ell_k(l)$. Moreover, we can say that any propositional formula f is reachable from some BS_i after k steps if $BS_i \models BS_P$ and $BS_i \models \ell_k(f)$. Since we propagate labels for literals, we examine the labels of literals essential to the formula to get the label of the formula. The label of a propositional formula f at level k, $\ell_k(f)$, is defined recursively as:

$$\ell_k(f \wedge f') = \ell_k(f) \wedge \ell_k(f')$$
$$\ell_k(f \vee f') = \ell_k(f) \vee \ell_k(f')$$
$$\ell_k(\neg(f \wedge f')) = \ell_k(\neg f \vee \neg f')$$
$$\ell_k(\neg(f \vee f')) = \ell_k(\neg f \wedge \neg f')$$
$$\ell_k(\top) = BS_P$$
$$\ell_k(\bot) = \bot$$

The LUG is based on IPP's [Koehler, 1999] planning graph, where there are three parts of a planning graph level: the action layer, effect layer, and literal layer. The extensions are to add label annotations to label the elements of the action $\mathcal{A}$, effect relation $\mathcal{E}$, and literal $\mathcal{L}$ layers. In the following discussion we formally define how to construct

[5] We refer to reachability in terms of optimistic reachability. As is the case with normal planning graphs, a set of literals that appears reachable in the planning graph may not be reachable in the state space because of undiscovered mutexes.

the LUG, extract a conformant relaxed plan to support a formula from all states of a belief state, BS_i, and how to post-process the conformant relaxed plan to form a contingent relaxed plan.

Label Propagation: Recall that a label is a formula describing a set of states (which is a subset of BS_P) from which a graph element is reachable. For example, if we build the LUG for $BTCS$, using $BS_P = BS_I$, we could say that $\neg arm$ is reachable from BS_I after one step if $\mathcal{L}_1$ contains $\neg arm$ labelled as $\ell_1(\neg arm) = BS_I$. The propagation of labels is based on the intuition 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. We now describe label propagation by showing how a graph level $\{\mathcal{A}_k, \mathcal{E}_k, \mathcal{L}_{k+1}\}$ is built, with respect to a belief state BS_P.

The LUG for $BTCS$, shown in Figure 2,using $BS_P = BS_I$ has the following initial layer:

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

Once the previous literal layer $\mathcal{L}_k$ is computed, we compute the labelled action layer $\mathcal{A}_k$. $\mathcal{A}_k$ contains all applicable causative actions from the action set A, plus all literal persistence, l_p.[6] An action's executability precondition, ρ_e, must be reachable for some possible world of BS_P at level k for the action to be applicable (i.e. $\ell_k(\rho_e) \not\models \bot$). If applicable, the action's label at level k, is:

$$\ell_k(a) = \ell_k(\rho_e)$$

The zeroth action layer for $BTCS$, is:

$$\begin{aligned}\mathcal{A}_0 = \{&\ell_0(DunkP1) = \ell_0(DunkP2) = \ell_0(Flush) = BS_P,\\ &\ell_0(\neg inP2_p) = \ell_0(inP1_p) = \ell_0(inP1),\\ &\ell_0(\neg inP1_p) = \ell_0(inP2_p) = \ell_0(inP2),\\ &\ell_0(clog_p) = \ell_0(\neg arm_p) = \bot,\\ &\ell_0(\neg clog_p) = \ell_0(arm_p) = BS_P\}\end{aligned}$$

The effect relations, $\mathcal{E}_k$, depend both on the literal layer, $\mathcal{L}_k$, and action layer, $\mathcal{A}_k$. An effect relation φ_i is applicable when the associated action is applicable and the antecedent of the effect, ρ_i is reachable for some possible world of BS_P (i.e. $\ell_k(\rho_i) \not\models \bot$). The label at k is the conjunction of the label of the associated action with the label of the formula of the effect's antecedent:

$$\ell_k(\varphi_i) = \ell_k(a) \wedge \ell_k(\rho_i)$$

The zeroth effect layer for $BTCS$, is:

$$\begin{aligned}&\mathcal{E}_0 =\\ &\{\ell_0(DunkP1 : \varphi_0) = \ell_0(DunkP2 : \varphi_0) = \ell_0(Flush : \varphi_0) = BS_P\\ &\ \ell_0(DunkP1 : \varphi_1) = \ell_0(\neg inP2_p : \varphi_0) = \ell_0(inP1_p : \varphi_0) = \ell_0(inP1),\\ &\ \ell_0(DunkP2 : \varphi_1) = \ell_0(\neg inP1_p : \varphi_0) = \ell_0(inP2_p : \varphi_0) = \ell_0(inP2),\\ &\ \ell_0(clog_p : \varphi_0) = \ell_0(\neg arm_p : \varphi_0) = \bot,\\ &\ \ell_0(\neg clog_p : \varphi_0) = \ell_0(arm_p : \varphi_0) = BS_P,\\ &\}\end{aligned}$$

[6] Persistence for a literal l, denoted by l_p, is represented as an action where $\rho_e = \varepsilon_{0,0} = l$.

The literal layer, $\mathcal{L}_k$, contains all literals. The label of a literal, $\ell_k(l)$, depends on $\mathcal{E}_{k-1}$ and is constructed as the disjunction of the labels of each effect that gives the literal. We say an effect φ_i gives a literal l when there exists a consequent outcome $\varepsilon_{i,j}$ of the effect that entails l:

$$\ell_k(l) = \bigvee_{\substack{\exists_j \varepsilon_{i,j} \models l,\\ \varphi_i \in \mathcal{E}_{k-1}}} \ell_{k-1}(\varphi_i)$$

The first literal layer for $BTCS$ is:

$$\begin{aligned}\mathcal{L}_1 = \{&\ell_1(inP1) = \ell_1(\neg inP2) = \ell_0(\neg inP2),\\ &\ell_1(\neg inP1) = \ell_1(inP2) = \ell_0(inP2),\\ &\ell_1(clog) = \ell_1(\neg clog) = \ell_1(\neg arm) = \ell_1(arm) = BS_P\}\end{aligned}$$

In our $BTCS$ example, level off occurs at level two, because the labels do not change between literal levels one and two.

When level off occurs, we can say that for any BS_i, where $BS_i \models BS_P$, that a formula f is reachable 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, back-chaining from k, to support f with respect to BS_i, described in the next section.

We can say that the goal in our example is reachable after two steps because $BS_P = BS_I \models \ell_1(\neg arm) = BS_P$.

Global Planning Graphs: We previously alluded to using different choices for our belief state, BS_P, from which we construct the LUG. Following previous work in progression search, an obvious choice for BS_P is the BS_i of a node for which we want to obtain a heuristic, an approach that builds a LUG for each search node. However, we note that we can define BS_P as a larger set of states than just BS_i, namely a set containing every reachable state in the problem. We find the reachable states by looking at the last level of a non-labelled planning graph built from all literals in the initial belief state. The advantage of defining BS_P this way is that we only have to build the LUG *once for the entire search episode*. Our conformant relaxed plan procedure does not rely on BS_P, aside from having $BS_i \models BS_P$ (which is assured for all reachable BS_i if BS_P represents all reachable states). We call this approach a Global LUG because we use a LUG to symbolically represent every possible planning graph that progression search may need.

Conformant Relaxed Plans

The relaxed plan heuristic we extract from the LUG is similar to the multiple graph relaxed plan heuristic, h^{MG}_{RPU}, described in [Bryce and Kambhampati, 2004]. The h^{MG}_{RPU} heuristic uses several planning graphs, one for every possible world of our source belief state BS_i, and extracts a relaxed plan from each to achieve a state of BS_G. It then unions the set of actions chosen at each step in each of the

relaxed plans to account for the overlap (positive interaction) of achieving subgoals the same way in multiple worlds.

The LUG conformant relaxed plan heuristic, h^{LUG}_{CRP}, is similar in that it counts actions that are applicable in multiple worlds only once and accounts for actions that are used in different subsets of the possible worlds. The advantage of h^{LUG}_{CRP} is that we find these actions by looking at only one planning graph, and extracting a single, albeit more complicated, relaxed plan. In the relaxed plan we want to support the goal with every state in BS_i, but in doing so we need to track which states in BS_i use which lines of causal support. In a classical planning relaxed plan, a line of causal support for a subgoal only refers to one world. But here multiple worlds may share the same line of support. Furthermore, a subgoal may not have a single line of support from all worlds in BS_i so multiple lines of support may be needed, each contributing support from a different set of worlds. The complications, and notation, arise from tracking what worlds use which lines of causal support to support subgoals.

Our conformant relaxed plans, $CRP_{BS_P}(BS_i, BS_G)$ are labelled subgraphs of a LUG, represented as a set of layers: $\{\mathcal{A}^{CRP}_0, \mathcal{E}^{CRP}_0, \mathcal{L}^{CRP}_1, ..., \mathcal{A}^{CRP}_{k-1}, \mathcal{E}^{CRP}_{k-1}, \mathcal{L}^{CRP}_k\}$, where $\mathcal{A}^{CRP}_r$ is a set of labelled actions, $\mathcal{E}^{CRP}_r$ is a set of labelled effects, and $\mathcal{L}^{RP}_{r+1}$ is a set of labelled clauses. We build the LUG in terms of literals, but extract a relaxed plan in terms of clauses because our goal may be expressed in terms of a formula. The actions, effects, and clauses that make up the relaxed plan are labelled to indicate the worlds where they are chosen for support.

For instance the relaxed plan, shown in bold in Figure 2, for BS_I to reach BS_G in $BTCS$ is:

$$
\begin{array}{l}
CRP_{BS_I}(BS_I, BS_G) = \\
\{\mathcal{A}^{CRP}_0 = \{\ell^{CRP}_0(DunkP1) = (arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), \\
\qquad \ell^{CRP}_0(DunkP2) = (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2)\}, \\
\mathcal{E}^{CRP}_0 = \{\ell^{CRP}_0(DunkP1 : \varphi_1) = (arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), \\
\qquad \ell^{CRP}_0(DunkP2 : \varphi_1) = (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2)\}, \\
\mathcal{L}^{CRP}_1 = \{\ell^{CRP}_1(\neg arm) = BS_I\}\}
\end{array}
$$

We already found that BS_G is reachable at level one, so we form $\mathcal{L}^{CRP}_1$ with the clauses of $\kappa(BS_G)$. The relevant effects from $\mathcal{E}_0$ are the φ_1 of $DunkP1$ and $DunkP2$, because they contribute support for $\neg arm$ in the worlds we care about. We need both effects to cover $\neg arm$ because $\ell_0(DunkP2 : \varphi_1) \vee \ell_0(DunkP2 : \varphi_1) \models \ell^{CRP}_1(\neg arm)$, so we have two lines of causal support, each contributing support in a world. We then insert the associated actions into the action layer. We can stop here, because we've covered the clauses in $\mathcal{L}^{CRP}_1$. Our action layer contains two actions, so the relaxed plan gives a heuristic value of two to support $\neg arm$ from BS_I.

For the general case, extraction starts at the level k where BS_G is first reachable from BS_i. The first relaxed plan layers we construct are $\mathcal{A}^{CRP}_{k-1}, \mathcal{E}^{CRP}_{k-1}, \mathcal{L}^{CRP}_k$, where $\mathcal{L}^{CRP}_k$ contains all clauses C, where $C \in \kappa(BS_G)$, labelled as $\ell^{CRP}_k(C) = BS_i$. Formally, we define the k^{th} clause layer as:

$$\mathcal{L}^{CRP}_k = \{\ell^{CRP}_k(C) = BS_i | C \in \kappa(BS_G)\}$$

For each level r, $1 \leq r \leq k$, we support each clause in $\mathcal{L}^{CRP}_r$ by choosing relevant effects from $\mathcal{E}_{r-1}$ to form $\mathcal{E}^{CRP}_{r-1}$. An effect φ_i is relevant if it covers some of the worlds where we need to cover C, namely $\ell_{r-1}(\varphi_i) \wedge \ell^{CRP}_r(C) \not\models \bot$ and it has a consequent outcome j that entails a literal in C, $\exists_{j, l \in C} \varepsilon_{i,j} \models l$.

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 $\mathcal{E}^{CRP}_{r-1}$ and labelled with the new worlds it covered for C, denoted by f_{φ_i} in the equation below. The label of an effect denotes for each clause C the worlds in which it was used to cover the clause, denoted by $f^C_{\varphi_i}$, and not the worlds in which other effects covered C, denoted by $f^C_{\overline{\varphi}_i}$. The effect layer $\mathcal{E}^{CRP}_{r-1}$ is defined as:

$$
\begin{array}{l}
\mathcal{E}^{CRP}_{r-1} = \{\ell(\varphi_i) = f_{\varphi_i} | \exists_{C \in \mathcal{L}^{CRP}_r, j, l \in C} \varepsilon_{i,j} \models l, f_{\varphi_i} \not\models \bot\} \\
f_{\varphi_i} = \bigvee_{\substack{C \in \mathcal{L}^{CRP}_r, \\ \exists_{j, l \in C} \varepsilon_{i,j} \models l}} f^C_{\varphi_i} \wedge \neg f^C_{\overline{\varphi}_j} \wedge BS_i \\
f^C_{\varphi_i} = \ell^{CRP}_r(C) \wedge \ell_{r-1}(\varphi_i) \\
f^C_{\overline{\varphi}_j} = \bigvee_{\substack{\varphi'_i \in \mathcal{E}^{CRP}_{r-1}, \\ \varphi_i \neq \varphi'_i, \\ \exists_{j, l \in C} \varepsilon_{i,j} \models l, \\ \exists_{j', l' \in C} \varepsilon'_{i,j'} \models l'}} \ell^{CRP}_{r-1}(\varphi'_i)
\end{array}
$$

An additional requirement on $\mathcal{E}^{CRP}_{r-1}$ is that it in fact covers all worlds of each clause in $\mathcal{L}^{CRP}_r$:

$$\forall_{C \in \mathcal{L}^{CRP}_r} \ell^{CRP}_k(C) \models \bigvee_{\substack{\varphi_i \in \mathcal{E}^{CRP}_{r-1}, \\ \exists_{j, l \in C} \varepsilon_{i,j} \models l}} \ell^{CRP}_r(\varphi_i)$$

Once all clauses in $\mathcal{L}^{CRP}_r$ are covered, we form the action layer $\mathcal{A}^{CRP}_{r-1}$ as all actions that have an effect in $\mathcal{E}^{CRP}_{r-1}$. The actions in $\mathcal{A}^{CRP}_{r-1}$ are labelled to indicate all worlds where any of their effects were labelled in $\mathcal{E}^{CRP}_{r-1}$:

$$
\begin{array}{l}
\mathcal{A}^{CRP}_{r-1} = \{\, \ell^{CRP}_{r-1}(a) = f_a \,|\, \exists_{\varphi_i \in \Phi_a} \varphi_i \in \mathcal{E}^{CRP}_{r-1}\} \\
f_a = \bigvee_{\substack{\varphi_i \in \Phi_a, \\ \varphi_i \in \mathcal{E}^{CRP}_{r-1}}} \ell^{CRP}_{r-1}(\varphi_i)
\end{array}
$$

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

$$
\begin{array}{l}
\mathcal{L}^{CRP}_{r-1} = \left\{ \ell^{CRP}_{r-1}(C) = f_C \;\middle|\; \begin{array}{l} 1 < r < k \text{ and,} \\ (\exists_{a \in \mathcal{A}^{CRP}_{r-1}} C \in \kappa(\rho_e) \text{ or,} \\ \quad \exists_{\varphi_i \in E^{CRP}_{r-1}} C \in \kappa(\rho_j)) \end{array} \right\} \\
f_C = \bigvee_{\substack{a \in \mathcal{A}^{CRP}_{r-1}, \\ C \in \kappa(\rho_e)}} \ell^{CRP}_{r-1}(a) \vee \bigvee_{\substack{\varphi_i \in \mathcal{E}^{CRP}_{r-1}, \\ C \in \kappa(\rho_j)}} \ell^{CRP}_{r-1}(\varphi_i)
\end{array}
$$

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

Once we get a conformant relaxed plan, $CRP_{BS_P}(BS_i, BS_G)$, that supports BS_G in all worlds of BS_i, we compute the conformant relaxed plan heuristic for $POND$ as:

$$h^{(G)LUG}_{CRP}(BS_i) = \sum_{r=0}^{k-1} \mid \mathcal{A}_r^{CRP} \mid$$

This conformant relaxed plan is estimating a strong plan because at each level we ensure that the chosen actions will support the subgoals from all possible worlds where they are needed to support subgoals at the next level. In many cases the relaxed plan can use one action to support subgoals in several possible worlds. This is useful in guiding the search towards plans with lower overall plan length and higher positive interaction in achieving the goal from all possible worlds.

Contingent Relaxed Plans

Up until now we have not explicitly considered sensory actions in our heuristics. A straightforward generalization of our approach may consider extending the ideas in SGP[Weld *et al.*, 1998]. SGP tracks reachable partitions of worlds, i.e. worlds that can be disambiguated from others via sensing. Building a labelled version of the SGP planning graphs would be costly because it is hard to compactly represent these partitions in the LUG. Instead of propagating all possible partitions of possible worlds, we use a phased relaxation approach. We build a LUG and extract a conformant relaxed plan, then insert sensors into the conformant relaxed plan. Inserting a sensor means that we create new contexts (branches) out of the context in which it is applied. We can determine how a sensor will partition a context by looking at the literals in the LUG – effectively computing partitions on-demand. Having contexts in our relaxed plan allows us to reason about which worlds must execute which lines of causal support. Our intention is to capture how sensory actions i) reduce the overall expected cost of a plan, and ii) increase immediate cost by requiring set-up.

We extend our representation of conformant relaxed plans to contingent relaxed plans $SRP_{BS_P}(BS_i, BS_G)$ by adding partitions, $\mathcal{P}_r^{SRP}$, to each level, so a contingent relaxed plan has the structure $\{\mathcal{A}_0^{SRP}, \mathcal{P}_0^{SRP}, \mathcal{E}_0^{SRP}, \mathcal{L}_1^{SRP}, ..., \mathcal{A}_{k-1}^{SRP}, \mathcal{P}_{k-1}^{SRP}, \mathcal{E}_{k-1}^{SRP}, \mathcal{L}_k^{SRP}\}$. A partition, $\mathcal{P}_r$, for a level r contains several context formulas p. The context formulas denote the worlds of our belief state BS_i that we can isolate with sensory actions and observations. Each context p is seen as the subset of the states in BS_i that will enter a particular branch of the relaxed plan. A branch of the relaxed plan is defined on a per level basis by a context p and the actions in $\mathcal{A}_r^{SRP}$ where $p \wedge \ell_r^{SRP}(a) \not\models \bot$. The value of our sensory relaxed plan is the sum of the expected context costs of each level. The expected context cost of a level is simply the sum of the number of actions in each context divided by the number of contexts.

We construct our sensory relaxed plan by extracting a conformant relaxed plan, as previously described, and then process it to include sensory actions. As outlined in Figure 1, we go through each level of the conformant relaxed plan,

Process(BS_i, CRP, A):
1: $SRP = CRP$
2: $r = 0$
3: **while** $r < max_j(A_j^{RP})$, **do**
4: **if** $r = 0$, **then**
5: $\mathcal{P}_r^{SRP} = \{BS_i\}$
6: **else**
7: $\mathcal{P}_r^{SRP} = \mathcal{P}_{r-1}^{SRP}$
8: **endif**
9: **for** each sensory a in A , **do**
10: **for** each partition $p \in \mathcal{P}_r^{SRP}$ s.t. $p \models \ell_r(\rho_e)$, **do**
11: $\mathcal{P}'$ =Split(p, a, r)
12: **if** $|\mathcal{P}'| > 1$, **then**
13: **for** each $\langle p', o_j \rangle \in \mathcal{P}'$, **do**
14: $CRP' = CRP_{BS_P}(p', o_j)$
15: $SRP = SRP \cup CRP'$
16: **endfor**
17: $CRP' = CRP_{BS_P}(p, \rho_e)$
18: $SRP = SRP \cup CRP'$
19: $\mathcal{P}_r^{SRP} = (\mathcal{P}_r \cup_{\langle p', o_j \rangle \in \mathcal{P}'} p') \setminus p$
20: **endif**
21: **endfor**
22: **endfor**
23: r++
24: **endwhile**
25: **return** SRP

Figure 3: Sensory Relaxed Plan algorithm pseudo-code.

Split(p, a, r):
1: $I = \bigvee_{o_i, o_j, i \neq j} (\ell_r(o_i) \wedge \ell_r(o_j) \wedge p)$
2: $\mathcal{P}' = \{\langle p', o_j \rangle | p' = \ell_r(o_j) \wedge \neg I \wedge p$ and $p' \not\models \bot\}$
3: **for** every $\hat{S}$ in $\hat{\xi}(p \wedge \neg I)$, **do**
4: pick $\langle p, o_j \rangle \in \mathcal{P}'$ s.t. $min_k(\hat{S} \models \ell_k(o_j))$
5: set $\langle p, o_j \rangle = \langle p \vee \hat{S}, o_j \rangle$
6: **endfor**
7: **return** $\mathcal{P}'$

Figure 4: Split algorithm pseudo-code.

adding sensors and computing partitions. Our initial partition, $\mathcal{P}_0^{SRP}$, contains one context, BS_i. We then try to split the BS_i context with every sensory action whose executability preconditions are reachable (line 11). The splitting of a context with a sensory action a may give a set $\mathcal{P}'$ of new contexts matched with each observation of Φ_a, denoted as $\langle p', o_j \rangle$. We make sure that at least two new contexts are made, otherwise the sensory action is of no use. A sensory action may fail to create new contexts if for example only one of its observations is reachable. We then find a conformant relaxed plan to support each observation o_j in the worlds denoted by p' because we need to make sure the observation can be made. The conformant relaxed plan is then unioned with our sensory relaxed plan to have the contingent relaxed plan reflect causal support for an observation. We also find conformant relaxed plans for the executability preconditions of the sensory action and union it with the contingent relaxed plan. Finally, we remove the old context, p, from $\mathcal{P}_r^{SRP}$ and add the new contexts to $\mathcal{P}_r^{SRP}$. Since we may split every p in $\mathcal{P}_r^{SRP}$ and add new contexts, the for loop (lines 9-22) is a fix-point computation. Reaching the fix-point makes us move to the next level, copying over the contexts we just made and trying to make more.

Splitting a context p (Figure 4) with a sensory action a at level r involves finding the worlds of p where each outcome o_j of the sensory action is reachable. The set of worlds described by the intersection, I, of worlds where every o_j is

Problem	POND h^{MG}_{RPU}	POND h^{LUG}_{CRP}	POND h^{GLUG}_{CRP}	MBP	KACMBP	HSCP	GPT	CGP	SGP	CFF
$Rovers1$	370/**5**	360/**5**	260/**5**	66/**5**	9293/**5**	OoM	3139/**5**	70/**5**	70/**5**	6/**5**
2	1440/**9**	750/**8**	320/**8**	141/**8**	9289/**15**	-	4365/**8**	180/**8**	30/**8**	6/**8**
3	7260/**11**	1250/**10**	400/**10**	484/**10**	9293/**16**	-	5842/**10**	460/**10**	1750/**10**	10/**10**
4	OoM	1790/**13**	510/**13**	570/**15**	9371/**18**	-	7393/**13**	1860/**13**	TO	10/**13**
5	-	9750/**24**	2250/**24**	OoM	39773/**40**		399525/**20**	TO	-	18/**22**
6	-	58800/**23**	13750/**23**	727/**32**	TO	-	TO	-	-	21/**23**
$Logistics1$	520/**9**	450/**9**	210/**9**	37/**9**	127/**12**	352/**9**	916/**9**	60/**6**	70/**6**	10/**9**
2	6100/**19**	2200/**15**	470/**15**	486/**24**	451/**19**	OoM	1297/**15**	290/**6**	510/**6**	12/**15**
3	13260/**14**	4320/**14**	810/**14**	408/**14**	1578/**18**	-	1711/**11**	400/**8**	4620/**8**	14/**12**
4	OoM	10580/**18**	2390/**18**	2881/**27**	8865/**22**	-	9828/**18**	1170/**8**	447470/**8**	12/**18**
5	-	30930/**28**	8150/**28**	OoM	226986/**42**	-	543865/**28**	TO	TO	25/**28**
$BTC10$	1080/**19**	250/**19**	200/**19**	504/**19**	45/**19**	25/**19**	715/**19**	39370/**19**	TO	57/**19**
30	OoM	12120/**59**	6540/**59**	268/**59**	635/**59**	293/**59**	-	-	-	23629/**59**
50	-	104440/**99**	75190/**99**	1287/**99**	10821/**99**	1352/**99**	-	-	-	334879/**99**
70	-	588630/**139**	457890/**139**	3625/**139**	9334/**139**	OoM	-	-	-	-
Cube 5	OoM	630/**12**	290/**15**	25/**24**	16/**12**	29/**12**	82/**12**	TO	TO	21/**12**
11	-	33110/**31**	12440/**30**	215/**60**	31/**30**	566/**30**	308875/**30**	-	-	264/**30**
17	-	329010/**48**	219850/**48**	1154/**96**	70/**48**	3598/**48**	TO			2957/**48**
CubeC 5	OoM	3210/**20**	730/**18**	16/**18**	18/**18**	418/**18**	362/**18**	TO	TO	36305/**45**
7	-	9570/**31**	6310/**29**	35/**27**	29/**27**	4557/**27**	4781/**27**	-	-	TO
9	-	177890/**40**	11080/**40**	64/**36**	70/**36**	18896/**36**	42257/**36**	-	-	-
Ring 3	1350/**10**	360/**10**	290/**10**	8/**8**	8/**8**	12/**8**	568/**8**	TO	TO	TO
4	22720/**14**	3890/**15**	1870/**15**	21/**11**	18/**11**	20/**11**	605/**11**	-	-	-
5	OoM	85490/**17**	43800/**17**	33/**14**	33/**14**	37/**14**	1186/**14**	-	-	-
6	-	596430/**23**	268800/**23**	74/**17**	66/**17**	96/**17**	51469/**17**	-	-	-

Figure 5: Results for $POND$ using h^{MG}_{RPU}, h^{LUG}_{CRP}, and h^{GLUG}_{CRP}, MBP, KACMBP, HSCP, GPT, CGP, SGP, and CFF for conformant $Rovers$, $Logistics$, BTC, $Cube$, $CubeC$, and $Ring$. The data is Total Time (ms) / # plan steps (in bold), "-" indicates no attempt, OoM represents out of memory, and TO represents a time out.

reachable must be divided among the o_j in a splitting. The worlds $\neg I \wedge \ell_r(o_j) \wedge p$ can be safely assigned to each such o_j without overlapping with another o_i. Each world in the intersection is assigned to an o_j such that o_j is the first observation reachable in that world.

Once sensors are added to the relaxed plan, we can compute the cost of the contingent relaxed plan as the sum of the cost of the action levels. The cost of an action level is the sum of the cost of the contexts divided by the number of contexts. The cost of a context is the number of actions that can be executed in the context. An action can be executed in a context when the label of the action not inconsistent with the formula describing the context.

$$h^{(G)LUG}_{SRP} = \sum_{r=0}^{k-1} \left(\frac{\sum_{p \in P_r} \left| \left\{ a \;\middle|\; \begin{array}{l} a \in A^{SRP}_r, \\ \ell^{RP}_r(a) \wedge p \not\models \bot \end{array} \right\} \right|}{|P_r|} \right)$$

From our $BTCS$ example, we start with $\mathcal{P}_0 = \{BS_I\}$. We find that the executability preconditions of $DetectMetal$ are reachable at level zero, and we can use it to split BS_I. Splitting BS_I, we get $\mathcal{P}' = \{\langle (arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), o_0 \rangle, \langle (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2), o_1 \rangle\}$. The conformant relaxed plans to support ρ_e, o_0, and o_1 of $DetectMetal$ are empty because they hold at level zero, so our relaxed plan becomes:

$$\begin{array}{l} SRP_{BS_P}(BS_i, BS_G) = \\ \{\mathcal{A}^{SRP}_0 = \mathcal{A}^{CRP}_0 \cup \{\ell^{SRP}_0(DetectMetal) = BS_I\}, \\ \quad \mathcal{P}^{SRP}_0 = \{(arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), \\ \qquad (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2)\}, \\ \quad \mathcal{E}^{SRP}_0 = \mathcal{E}^{CRP}_0 \cup \\ \qquad \{\ell^{RP}_0(DetectMetal : o_0) = (arm \wedge \neg clog \wedge inP1 \wedge \neg inP2), \\ \qquad\; \ell^{RP}_0(DetectMetal : o_1) = (arm \wedge \neg clog \wedge \neg inP1 \wedge inP2)\}, \\ \quad \mathcal{L}^{SRP}_1 = \mathcal{L}^{CRP}_1\} \end{array}$$

Our heuristic value is two because each context has two actions – the appropriate dunk action and the sensory action – and there are two contexts, so ((1+2)/2)+((1+1)/2) = 2.5. This estimate is the same as our conformant relaxed plan, but in other domains the estimates can be very different.

Empirical Comparisons

We experimented with several existing domains and also developed two new domains to test our heuristic techniques. The existing domains include BTC (bomb in the toilet with clogging), $BTCS$ (bomb in the toilet with clogging and sensing), $CubeCorner$ ($Cube$), $CubeCenter$ ($CubeC$), $Ring$, and $Medical$ [Petrick and Bacchus, 2002]. We also developed conformant and contingent versions of the classical $Logistics$ and $Rovers$ domains. We also augment our contingent $Rovers$ domain to create three test sets with only contingent solutions and different numbers of preconditions for the available sensors.

We compare $POND$ to several conformant and contingent planners: MBP[Bertoli *et al.*, 2001a], KACMBP[Bertoli and Cimatti, 2002], HSCP[Bertoli *et al.*, 2001b], CGP[Smith and Weld, 1998], SGP[Weld *et al.*, 1998], GPT[Bonet and Geffner, 2000], and CFF[Brafman and Hoffmann, 2004]. We provide our planner binary, as well as the domain descriptions used for each planner at: http://rakaposhi.eas.asu.edu/belief-search. We tried our best to make the domain encodings standard across the planners by using binary fluents; there were a couple of instances with GPT where we had to use multi-valued fluents. The tests had a time out of 20 minutes and a memory limit of 1GB on a 2.8GHz P4 Linux machine. We also used a heuristic weight of $w = 5$ in $POND$. We note that our implementation makes extensive use of propositional entailment ($\models$) as well as the other common

Problem	POND h^{MG}_{RPU}	POND h^{LUG}_{CRP}	POND h^{GLUG}_{CRP}	POND h^{GLUG}_{SRP}	MBP	GPT	SGP
$Rovers1$	360/**5**	360/**5**	260/**5**	260/**5**	3312/**11**	3148/**5**	70/**5**
2	790/**9**	820/**7**	320/**7**	390/**8**	4713/**75**	5334/**7**	760/**7**
3	970/**10**	1370/**8**	380/**8**	490/**9**	5500/**119**	7434/**8**	TO
4	1490/**14**	1910/**10**	490/**10**	630/**12**	5674/**146**	11430/**10**	-
5	OoM	29600/**19**	4040/**19**	7180/**21**	16301/**76**	TO	-
6	-	65100/**19**	9080/**21**	12290/**20**	OoM	-	-
$Logistics1$	400/**8**	400/**7**	200/**7**	230/**7**	41/**16**	1023/**7**	5490/**6**
2	2150/**13**	3150/**12**	560/**12**	870/**12**	22660/**177**	5348/**12**	TO
3	2560/**10**	3760/**9**	700/**9**	970/**9**	2120/**45**	2010/**8**	-
4	31350/**17**	17870/**15**	2770/**15**	4840/**15**	OoM	TO	-
5	OoM	88810/**23**	14480/**23**	28170/**21**	OoM	-	-
$BTCS$ 20	20150/**20**	3820/**20**	1690/**20**	2930/**20**	OoM	TO	TO
40	OoM	52740/**40**	39770/**40**	62390/**40**	-	-	-
60	-	373460/**60**	344650/**60**	555600/**60**	-	-	-
$Medical$ 20	2360/**3**	2070/**3**	360/**3**	430/**3**	TO	92/**3**	TO
40	20500/**3**	47140/**3**	1930/**3**	2950/**3**	-	1079/**3**	-
60	OoM	287390/**3**	21660/**3**	29070/**3**	-	2524/**3**	-
80	-	TO	203210/**3**	208420/**3**	-	7869/**3**	-
100	-	-	496670/**3**	507990/**3**	-	15762/**3**	-

	$POND$ h^{GLUG}_{CRP}	$POND$ h^{GLUG}_{SRP}	MBP
0ep-$Rovers1$	420/**5.00**	510/**5.00**	3320/**11.00**
2	590/**6.50**	1110/**6.50**	4762/**65.00**
3	680/**5.75**	1310/**5.75**	5500/**87.00**
4	1000/**6.12**	1670/**6.00**	5674/**52.75**
5	12650/**15.53**	21780/**15.59**	16301/**56.44**
6	34460/**18.62**	63070/**17.37**	TO
5ep-$Rovers5$	420/**5.00**	520/**5.00**	3568/**22.00**
2	3470/**12.00**	1990/**11.50**	13754/**200.50**
3	5080/**17.00**	1260/**13.25**	26592/**225.66**
4	6580/**19.87**	2220/**14.75**	27750/**157.00**
5	31060/**31.81**	38020/**28.68**	40566/**214.75**
6	84730/**29.50**	216030/**26.25**	TO
10ep-$Rovers1$	770/**5.00**	1020/**5.00**	4416/**40.00**
2	TO	54740/**16.50**	296346/**341.50**
3	-	3240/**21.25**	TO
4	-	6800/**24.25**	-
5	-	TO	-
6	-	-	-

Figure 6: The first table shows results for $POND$ using h^{MG}_{RPU}, h^{LUG}_{CRP}, h^{GLUG}_{CRP}, and h^{GLUG}_{SRP}, MBP, GPT, and SGP for contingent $Rovers$, $Logistics$, $BTCS$, and $Medical$. The second table shows results for extra preconditions $Rovers$ (Xep-$Rovers$) using the h^{GLUG}_{CRP} and h^{GLUG}_{SRP} heuristics in $POND$ and MBP. The data is Total Time (ms) / # plan steps in maximum length branch (in bold), "-" indicates no attempt, OoM represents out of memory , and TO represents a time out.

propositional logic operations and using BDDs affords reasonable run-time performance.

We proceed by discussing how $POND$ performs in both conformant and contingent planning. Within $POND$ we compare the effectiveness of building the LUG for every search node (h^{LUG}_{CRP}) to building the LUG globally, (h^{GLUG}_{CRP}), as well as our improvements over building multiple planning graphs (h^{MG}_{RPU}) for each search node. We also evaluate the effectiveness of the contingent relaxed plan heuristic (h^{LUG}_{SRP}). Then, we discuss the relative advantages of $POND$ with respect to competing approaches.

Conformant Planning: Figure 5 shows results for the conformant domains. The first observation about $POND$ is that for all problems building the LUG globally, $GLUG$, has dramatic speed improvements over building the LUG at each search node, and both of these are significant improvements of the MG approach. We do not show results for the SRP heuristic because it reduces to the CRP heuristic in conformant planning.

In comparison to other planners, $POND$ performs reasonably well in many of the existing conformant planning domains, but is not the strongest planner in any domain. $POND$ is able to outperform MBP, KACMBP, and HSCP in the $Rovers$ and $Logistics$ domains by solving more problems, generally faster, and with typically higher quality solutions. $POND$ does typically better in terms of search time with respect to GPT in most of the domains. $POND$ out-scales CGP and SGP in all of the domains. CFF does much better in the $Logistics$ and $Rovers$ problems, but has more trouble in the BTC, $CubeC$, and $Ring$ domains.[7]

We note that $POND$ does as well as it does because of its effective heuristics despite its admittedly inefficient progression implementation. The reason $POND$ is able to do better than MBP in some problems is because our heuristics prove to be more important than efficient child generation.

Contingent Planning: The first table in figure 6 shows results for the contingent domains $Rovers$, $Logistics$, $BTCS$, and $Medical$. Again, $POND$ does better in all problems by using the $GLUG$ over the LUG, and both of these are improvements over the MG approach.

In comparison to the other planners, there are two observations: i) $POND$ exhibits scalability, and ii) $POND$ returns high quality plans. GPT only scales well on the $Medical$, and fails to find plans for large instances in other domains. SGP does not scale very well on any of the problems. MBP is able to solve many of the $Rovers$ and $Logistics$ instances, but at a cost of generating highly inoptimal plans, *an order of magnitude longer in some cases*, compared to $POND$, GPT and SGP.[8]

Costly Sensor Preconditions: The results shown in the second table in Figure 6 are for the $Rovers$ domain with extra sensory preconditions, using the conformant relaxed plan heuristic h^{GLUG}_{CRP} and the sensory relaxed plan heuristic h^{GLUG}_{SRP} in $POND$, and MBP. In this domain, each sensor has the original precondition of being at a particular waypoint to do the sensing, plus some number of additional preconditions (e.g. the camera being on, clean, focused, arm placed, etc.). Each additional precondition requires an action to re-establish it at each waypoint the rover navigates to. Each set of rows in the second table in Figure 6 is a sensor formulation with a different number of preconditions (0, 5, and 10). Each cell shows the total time to find a plan, and the solution plan's expected execution length. Here we change our measure of plan quality to expected length because it was much easier to cull from the compared planners' output. What we noticed is that: contingent relaxed plans (SRP) are useful in scaling the planner on problems with costly sensors, and tend to improve solution quality. We also noticed

[7]Our encoding of $Ring$ is especially difficult for CFF, and may be different from their encoding. We think it may have to do with extra literals in conditional effect antecedents.

[8]We compare maximal branch length rather than expected branch length because many of the planners do not provide the expected branch length.

that MBP has trouble scaling as sensors have additional preconditions and that MBP finds fairly bad plans, in terms of expected execution length.

Related Work

Although $POND$ utilizes planning graphs similar to CGP[Smith and Weld, 1998] and Frag-plan[Kurien *et al.*, 2002], in contrast to them, it only uses them to compute reachability estimates. The search itself is conducted in the space of belief states. $POND$ is 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]. $POND$ is similar to FF in that it uses progression search based on planning graph heuristics.

Another related line of work is in the MBP-family of planners—MBP [Bertoli *et al.*, 2001a], CMBP [Cimatti and Roveri, 2000], HSCP [Bertoli *et al.*, 2001b] and KACMBP [Bertoli and Cimatti, 2002]. Like $POND$, the MBP-family of planners all represent belief states in terms of binary decision diagrams, but vary with respect to the heuristics they implement and their search strategy.

More recently, there has been closely related work on heuristics for constructing conformant plans within the planner CFF [Brafman and Hoffmann, 2004]. The approach taken in CFF is to construct a SAT encoding to compute a relaxed plan heuristic. The SAT encoding obtains distance measures for size-2 disjunctions over literals in a belief state rather than all states in a belief state, as we do.

In contrast to these approaches, PKSPlan [Petrick and Bacchus, 2002] is a forward-chaining *knowledge-based planner* that requires a richer domain encoding. 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.

Conclusion

We described three important extensions to reachability heuristics for belief space–the labelled uncertainty graphs (LUG), the global labelled uncertainty graphs ($GLUG$) and contingent relaxed plans. We have implemented these on top of a progression planner for belief space planning called $POND$, and have shown that heuristics based on the LUG and $GLUG$ lead to significant scale-ups over those based on multiple planning graphs ones, and that contingent relaxed plans can help improve plan quality as well as scalability in problems with difficult to support sensory actions. We also reported on comparison studies with several state of the art conformant and contingent planners. These results show that $POND$ with our heuristics is competitive with the best conformant planners, and is able to produce significantly higher quality contingent plans. These results are particularly encouraging considering that our current implementation of $POND$ does not take advantage of the most efficient (BDD-based) child generation techniques, as in [Bertoli *et al.*, 2001a]. We are working on incorporating these improvements in the base planner, and expect that the experiments with the resulting planner will further improve our comparative performance.

Acknowledgements: We would like to thank Will Cushing and Dan Weld for helpful discussions and comments. This work was supported in part by the MCT/NASA 2004 summer internship program and NSF grants IIS-0308139.

References

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

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

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

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

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

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

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.

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

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

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

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.

Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer-Verlag, 1998.

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.

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

Jussi Rintanen. Expressive equivalence of formalisms for planning with sensing. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 2003.

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

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