State Agnostic Planning Graphs

and the application to belief-space planning

William Cushing and **Daniel Bryce**
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809
{william.cushing|dan.bryce}@asu.edu

Abstract

Planning graphs have been shown to be a rich source of heuristic information for many kinds of planners. In many cases, planners must compute a planning graph for each element of a set of states. The naive technique enumerates the graphs individually. This is equivalent to solving an all-pairs shortest path problem by iterating a single-source algorithm over each source.

We introduce a structure, the state agnostic planning graph, that directly solves the all-pairs problem for the relaxation introduced by planning graphs. The technique can also be characterized as exploiting the overlap present in sets of planning graphs. For the purpose of exposition, we first present the technique in classical planning. The more prominent application of this technique is in belief-space planning, where an optimization results in drastically improved theoretical complexity. Our experimental evaluation quantifies this performance boost, and demonstrates that heuristic belief-space progression planning using our technique is competitive with the state of the art.

Introduction

Heuristics derived from planning graphs (Blum & Furst 1995) are widespread in planning (Gerevini, Saetti, & Serina 2003; Hoffmann & Nebel 2001; Bonet & Geffner 1999; Younes & Simmons 2003; Nguyen, Kambhampati, & Nigenda 2002). In many cases, heuristics are derived from a set of planning graphs. For example, progression planners compute a planning graph (PG) for every search state. The same situation arises in belief space planning: one method of generating a heuristic for a belief is to build a planning graph for each member of the belief (Bryce & Kambhampati 2004).

A set of planning graphs for related states can be highly redundant. That is, any two planning graphs often overlap significantly. As an extreme example, the planning graph for a child state is a sub-graph of the planning graph of the parent, left-shifted by one step (Zimmerman & Kambhampati 2005). Computing a set of planning graphs by enumerating its members is, therefore, inherently redundant.

Motivating Example: Consider progression planning in a classical domain whose states consist of sets of letters and digits. So, "a1" and "ac45" are both states. Let there be operators to transform any letter into any other letter: for all letters α and β (not equal), there is an operator $o_{\alpha\beta}$=$(\{f_\alpha = \text{true}\}, \{f_\alpha = \text{false}, f_\beta = \text{true}\})$. For digits, let there be increment operators. For example, the shortest plan to reach "be15" from "ad04" uses 4 steps.

For this example, all of the planning graphs built will have much in common. In particular, consider the sub-graphs over letters. Each is nearly identical: using a single transition, any letter is reachable. So, in level 1, all letters and all letter manipulating operators are added to the graph. In terms of letters, then, there are 26 different sub-graphs, all with an identical level 1.

Contribution: In this paper, we develop an elegant generalization of the Planning Graph called the State Agnostic Graph (SAG). The SAG extends the labeling technique introduced in the work on the Labeled Uncertainty Graph (PG_{LUG}) (Bryce, Kambhampati, & Smith 2004). That work estimates belief-space distance by aggregating estimates of state-space distance. This is achieved, conceptually, by building a PG for each state contained in the belief. The PG_{LUG} is introduced as an efficient representation of this set of PGs. We take a more general perspective on the labeling technique employed, described here as SAG, and apply this perspective to further boost the performance of a belief-space progression planner, POND, employing PG_{LUG}-based heuristics.

We view the planning graph as exactly solving a single-source shortest path problem, for a relaxed planning problem. The levels of the graph efficiently represent a breadth-first sweep from the single source. In the context of progression planning, the planner will end up calculating a heuristic for many different sources. Iterating a single-source algorithm over each source (building a PG per search node) is a naive solution to the all-pairs shortest path problem. We develop our generalization, SAG, under this intuition: directly solving the all-pairs shortest path problem is more efficient than iterating a single source algorithm.

This intuition falls short in most planning problems, because the majority of states are unreachable. Reasoning about such states is useless, so instead, we develop the SAG as a solution to the multi-source shortest path problem. More precisely, the SAG is a representation of a set of PGs. Its exact form depends upon the kind of PG being generalized (mutexes, cost, time, ...). The main insight to the technique is to represent the propagation rules of the PG and a set of sources as boolean functions. Composing these functions via boolean algebra yields an approach for building the set of PGs without explicitly enumerating its elements. Manipulating formulas exploits redundant sub-structure, and this boosts empirical performance (Bryce, Kambhampati, & Smith 2004), in spite of the fact that this technique does not alter theoretical worst-case complexity.

Organization: We introduce the SAG technique, for the purpose of exposition, in the simpler realm of classical planning. We present a well-known kind of PG for classical planning, the Relaxed Planning Graph (PG_R). We generalize the PG_R to the Relaxed State Agnostic Graph (SA_R). We extend our discussion to belief-space planning, generalizing the PG_{LUG} to its state agnostic version: SA_{LUG}. We then demonstrate an equivalent formalization, $SLUG$, which drastically improves theoretical complexity: we use this result as the basis for an efficient belief-space progression planner, POND. From there, we consider several strategies for reducing redundant heuristic computations. Our experimental evaluation begins by comparing these strategies. We take the best among these, Reachable-SAG, to conduct an external comparison with state of the art belief space planners. We demonstrate that POND is competitive with the state of the art in both conformant and conditional planning. Before concluding, we provide some insight into the connections to other work.

Classical Planning

We first present SAG in a classical setting. We start with a formal definition of the kind of planning graph used in state of the art progression planners, PG_R. We touch upon the level heuristic ($h_{lev}^{PG_R}$), before moving on to present the state agnostic version of PG_R, SA_R. This graph allows us to extract PG-heuristics for a set of initial states. For the specific case of the level heuristic, we capture this in a formal definition ($h_{lev}^{SA_R}$).

Definition 1 (PG_R) *A Relaxed Planning Graph, $PG_R(s)=(V,E)$, built for a single source s, satisfies:*

1. *If f holds in s then $(f,0) \in V$*
2. *For any i such that $(f,i) \in V$ for every $f \in pre(o)$, then $(o,i) \in V$ and $((g,i),(o,i)) \in E$ for all $g \in pre(o)$*
3. *For any i such that $(o,i) \in V$, then $(f,i+1) \in V$ and $((o,i),(f,i+1)) \in E$ for all $f \in \mathit{eff}(o)$*

The intuition is that (x,k) is in the graph when it is feasible that x (a literal or operator) could be k-reachable under the full set of constraints of the planning problem. The *level* of x is simply the minimum i such that (x,i) is in the graph. The level heuristic is based on the extension of this idea to a set of literals (without mutexes, the level and max heuristics are equivalent).

Definition 2 ($h_{lev}^{PG_R}$) *The level heuristic gives an estimate of the distance between an initial state $\mathcal{I}$, and a goal $\mathcal{G}$. We restrict our attention to goals as conjunctions of literals.*

$$h_{lev}^{PG_R}(\mathcal{I},\mathcal{G}) = argmin_i \begin{array}{c} \forall x, (\mathcal{G} \models x) \Rightarrow \\ ((x,i) \in PG_R(\mathcal{I}).V) \end{array}$$

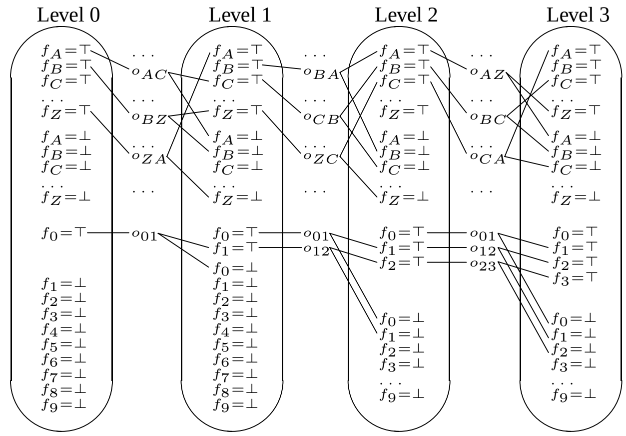

Figure 1: Graph structure of a SAG(labels omitted)

We generalize the PG_R to the State Agnostic Relaxed Planning Graph (SA_R), by permitting multiple source states. We associate every element of the graph with a label; the labels track which sources reach the associated graph element. These labels are boolean functions of the domain fluents. Intuitively, a source reaches a graph element if the label evaluates to true.

Definition 3 (SA_R) *A State Agnostic Relaxed Planning Graph is a labeled graph, $SA_R(S) = (V,E,l)$, with label function $l : V \rightarrow (\{\top,\bot\}^n \rightarrow \{\top,\bot\})$ assigning boolean functions to vertices, where n is the number of fluents–l assigns vertices a mapping from states to true and false. The graph is constructed from its scope S, a set of sources. For all $s \in S$, the following holds:*

1. *If f holds in s, then $l((f,0))(s)$ is true and $(f,0) \in V$*
2. *If $l((f,i))(s)$ is true for every $f \in pre(o)$, then:*
   - *(a) $l((o,i))(s)$ is true*
   - *(b) $(o,i) \in V$*
   - *(c) $((f,i),(o,i)) \in E$ for every $f \in pre(o)$*
3. *If $f \in \mathit{eff}(o)$ and $l((o,i))(s)$ is true, then $l((f,i+1))(s)$ is true and $(f,i+1) \in V$*

Consider building a SAG for the set of states that contain exactly one letter and only the digit zero. The first four levels are depicted in Figure 1. Each of the literals depicted in the first layer is present because it holds in one of the source states. For example, there is a single source state which contains the letter "d". Every other source state does not, so both f_d=$\top$ and f_d=$\bot$ are present at the first level. The remaining structure is constructed normally, except that elements with false labels are omitted.

The labels, not depicted in Figure 1, are built in step with the rest of the graph. The zeroeth level labels encode which of the source states reach the associated literal in zero steps; i.e., the subset of the scope that each literal holds in. We encode the scope, S, as a boolean formula:

$$\begin{array}{ll} S = (f_a \vee f_b \vee \cdots \vee f_z) \wedge & \text{(at least one letter)} \\ \neg(f_a \wedge f_b) \wedge \neg(f_a \wedge f_c) \ \wedge \cdots \wedge & \text{(at most one letter)} \\ \neg(f_b \wedge f_c) \wedge \neg(f_b \wedge f_d) \ \wedge \cdots \wedge & \vdots \\ \neg(f_y \wedge f_z) \wedge & \\ f_0 \wedge \neg f_1 \wedge \neg f_2 \wedge \cdots \wedge \neg f_9 & \text{(only zero)} \end{array}$$

From which one can form all of the label functions for the zeroeth level by conjoining with the appropriate projection function. For example: $l((f_d{=}\bot, 0))(s) = \neg f_d \wedge S$.

The remaining functions are built up from the initial labels. The function for an operator is the *and* of its preconditions' labels, likewise, the function for a literal is the *or* of its supporters' labels. Consider the operator o_{ad} at level 0. Its label is the conjunction of its single precondition, $f_a{=}\top$. So, $l((o_{ad}, 0)) = l((f_a, 0)) = f_a \wedge S$. The other 25 supporters of $f_d{=}\top$ are of the same form, so the function S can be factored out of the label of $f_d{=}\top$ at level 1: $l((f_d{=}\top, 1)) = (f_a \vee \cdots \vee f_z) \wedge S = S$. This simplifies down to just S because S implies that at least one letter is present. For the negative case, $f_d{=}\bot$, there is again a supporter for each letter, and so the label simplifies to S. A similar argument holds for the remaining letters; at all levels other than 0, the label of a letter literal is just S.

The digit literals only possess one distinct label function. The label of each at level 0 is either S or the literal is not present in the graph. Since all of the operators which affect digits precondition on only digits, whenever a digit literal is added to the graph it and all of its supporters are labeled with S.

Sharing: Unlike the PG, the SAG can be reused across multiple search states. Consider a set of sources, S. We extract a PG-heuristic for each source, $s \in S$, from $SA_R(S)$, *without building any additional graphs*. We do so by evaluating label functions. As an example, we present the formal generalization of $h^{PG_R}_{lev}$, $h^{SA_R}_{lev}$. We denote the sharing of the SAG across invocations of the heuristic by passing the graph as an argument. That is, the caller builds a graph for a set of anticipated search states, and passes that graph when obtaining the heuristic estimate for each search state in the set.

Definition 4 ($h^{SA_R}_{lev}$) *The graph, $G = SA_R(S)$, is built once for a set of states, S. For any $\mathcal{I} \in S$, the heuristic $h^{SA_R}_{lev}$ reuses the shared graph G.*

$$h^{SA_R}_{lev}(G, \mathcal{I}, \mathcal{G}) = argmin_i \begin{array}{c} \forall x, (\mathcal{G} \models x) \Rightarrow \\ ((x, i) \in G.V \wedge \\ G.l((x, i))(\mathcal{I})) \end{array}$$

Consider a goal, $\mathcal{G}{=}f_b \wedge f_5$, of making the state contain "b" and "5". Calculating a heuristic for a state, $\mathcal{I}$ = "a0", using $h^{PG_R}_{lev}(\mathcal{I}, \mathcal{G})$, requires building $PG_R(\mathcal{I})$. The cost of building this graph cannot be shared among search states. Even with early termination, any means of building this PG is bounded from below by the required 718 conclusions that it needs to draw. The majority of this quantity is the 26*25 reachable letter operators (the rest consists of 5 operators and 26*2 + 5*2+1 literals). Instead, we build $G{=}SA_R(\top)$ (for all states), depicted in Figure 2, and replace $h^{PG_R}_{lev}$ with $h^{SA_R}_{lev}$. Evaluating $h^{SA_R}_{lev}(G, \mathcal{I}, \mathcal{G})$, performs at most 2 label evaluations for "b" and 6 for "5", with the naive strategy of evaluating labels at successively increasing levels. Binary search improves this to 1 and 3 label evaluations, respectively. In our implementation of boolean formulas, evaluating a label requires no more than one operation per state fluent. In this example, the actual cost is much lower, but let us grossly estimate the cost as 8*36=388. We must also account for the cost of building G. Since G can be shared across every search state, we amortize the cost by the number of visited states. We conclude that there is quite some room for a performance boost: 388 + amortization is a loose upper bound on our technique, while 718 is a loose lower bound on the standard approach.

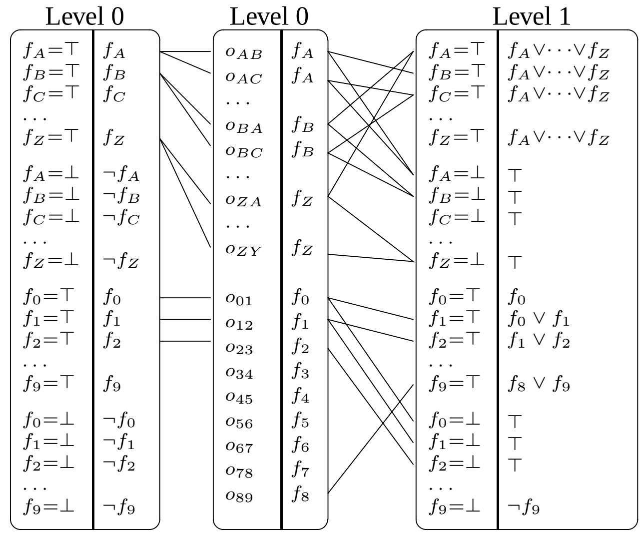

Figure 2: A SAG built for all states

Belief-Space Planning

We apply SAG to belief-space planning. Our presentation parallels the discussion in the classical setting: we formally present a PG-variant for belief-space planning, PG_{LUG}, and generalize to the state agnostic version, SA_{LUG}. The worst-case complexity of the SAG technique is exponential in the worst-case complexity of the PG method. That is, normally, there is no complexity-theoretic advantage to the technique. In the case of SA_{LUG}, we find an equivalent representation, $SLUG$ (see below), with the same worst-case complexity as PG_{LUG}–an exponential reduction.

We describe the operators in belief-space planning as tuples of deterministic conditional effects: $o^i = (o^{i1}, o^{i2}, \ldots, o^{ii_k})$, with $pre(o^{ij})$ and $eff(o^{ij})$ as antecedent and consequent, respectively, of conditional effect o^{ij} (Rintanen 2003). We extend our running example to belief-space by reinterpreting its operators: the executability precondition is removed, and the effect is conditioned on the former executability precondition.

Definition 5 (PG_{LUG}) *A Labeled Uncertainty Graph, $PG_{LUG}(b) = (V, E, l)$, is a labeled graph, built for the belief state b. The label, l, is a function from vertices of the* *graph to boolean functions of world states. For every $s \in b$, the following holds:*

1. *If f holds in s, then $l((f,0))(s)$ is true and $(f,0) \in V$*
2. *If $l((f,i))(s)$ is true for every $f \in pre(o^{xy})$, then:*
   (a) *$l((o^{xy},i))(s)$ is true*
   (b) *$(o^{xy},i) \in V$*
   (c) *$((f,i),(o,i)) \in E$ for every $f \in pre(o^{xy})$*
3. *If $f \in eff(o^{xy})$ and $l((o^{xy},i))(s)$ is true, then $l((f,i+1))(s)$ is true and $(f,i+1) \in V$*

As stated, the PG_{LUG} is a kind of SAG. The PG_{LUG} is an efficient representation of a set of PGs built for classical planning with conditional effects: the kind of PG is PG_{IPP} (Koehler 1999).[1] That is, the structure of the PG_{LUG} is simply the state agnostic generalization of PG_{IPP}: $PG_{LUG} = SA_{IPP}$. However, the PG_{LUG} is more than that: the PG_{LUG} is a full-fledged PG for belief-space planning. That is, Bryce, Kambhampati, & Smith derive heuristics for belief-space from it. We reproduce a formal description of $h^{PG_{LUG}}_{lev}$, extensions of other standard PG-heuristics are similar (Bryce, Kambhampati, & Smith 2004).

Definition 6 ($h^{PG_{LUG}}_{lev}$) *The function $h^{PG_{LUG}}_{lev}(b,\mathcal{G})$ is an estimate of the belief-space distance between the belief b and the goal G; it is an extension of the level heuristic.*

$$h^{PG_{LUG}}_{lev}(b,\mathcal{G}) = argmin_i \quad \begin{array}{l} \forall x, (\mathcal{G} \models x) \Rightarrow \\ \quad ((x,i) \in PG_{LUG}(b).V \wedge \\ \quad b \models PG_{LUG}(b).l((x,i))) \end{array}$$

Note that $h^{PG_{LUG}}_{lev}$ is a heuristic similar in structure to both $h^{PG_R}_{lev}$ and $h^{SA_R}_{lev}$. The heuristic does not amortize graph construction effort across search states, but, it does use the SAG technique to efficiently build and reason about a set of PGs. That is, the check "$b \models PG_{LUG}(b).l((x,i))$" reasons about a set of PGs (one for each $s \in b$) using only the single graph $PG_{LUG}(b)$.

We introduced Figure 1 as an example of SA_R. We re-interpret it as an example PG_{LUG}. The graph, $G = PG_{LUG}(b)$, depicted in Figure 1 is built for the belief, b, that one is certain that the state contains 0, and certain that the state contains exactly one letter.

We generalize PG_{LUG} to SA_{LUG}, by analogy with generalizing PG_R to SA_R. We introduce a label function to track which sources reach vertices. In this case, sources are beliefs. A further complication arises in that the PG_{LUG} already defines its own label functions. In order to complete the generalization, we introduce further propositions to capture these label functions. That is, we use two sets of propositions to define our label function. The first set of propositions, g_α, correspond to world states. For example, g_{ac45} represents the state "ac45": if true, then "ac45" is a member of the current belief. The second set of propositions are simply the domain fluents, f_α. The former set allows us to express properties of beliefs which the propagation rules of PG_{LUG} could depend upon. The latter set allow us to capture the additional data computed by PG_{LUG}.

[1] our discussion omits factoring out executability preconditions and unconditional effects; our implementation does not, and so possesses the 3-layer form of PG_{IPP}

Definition 7 (SA_{LUG}) *The structure $SA_{LUG}(B) = (V,E,l)$ is a labeled graph built for a set of belief states B. The label function is l : Vertex $\rightarrow$ ((Belief $\times$ State) $\rightarrow$ $\{\top,\bot\}$). The following holds for every $b \in B$, and $s \in b$:*

1. *If f holds in s, then $l((f,0))(b,s)$ is true and $(f,0) \in V$*
2. *If $l((f,i))(b,s)$ is true for every $f \in pre(o^{xy})$, then:*
   (a) *$l((o^{xy},i))(b,s)$ is true*
   (b) *$(o^{xy},i) \in V$*
   (c) *$((f,i),(o,i)) \in E$ for every $f \in pre(o^{xy})$*
3. *If $f \in eff(o^{xy})$ and $l((o^{xy},i))(b,s)$ is true, then $l((f,i+1))(b,s)$ is true and $(f,i+1) \in V$*

Intuitively, the label of a literal, f, at level k represents a set of (b,s) pairs. The label evaluates to true for some pair (b,s) if, and only if, $PG_{LUG}(b)$ considers the literal k-reachable from state s. As a concrete example, the label of $f_9 = \top$ at level 0 is:

$$l((f_9=\top,0))(b,s) = f_9 \wedge \left(\begin{array}{l} g_{a9} \vee g_{b9} \vee \ldots \vee g_{z9} \vee \\ g_{ab9} \vee \ldots \vee g_{a-z9} \vee \\ g_{a89} \vee \ldots \vee g_{a-z0-9} \vee g_9 \end{array} \right)$$

That is, the PG_{LUG} built for any belief which includes at least one state containing '9' considers $f_9 = \top$ a 0-reachable literal from any state where f_9 is true.

We optimize SA_{LUG} by eliminating the propositions g_α. Introducing these propositions is sufficient for representing arbitrary extensions of the PG to belief space. The PG_{LUG}, however, does not require this mechanical scheme. Intuitively, the propagation rules of the PG_{LUG} depend only upon properties of world states (as opposed to properties of beliefs). $SLUG$ exploits this: $SLUG(B)$ represents a set ($PG_{LUG}(b), b{\in}B$) for the price of a single element ($PG_{LUG}(b^*)$).

Definition 8 ($SLUG$) *An, optimized, State Agnostic Labeled Uncertainty Graph, $SLUG(B) = (V,E,l)$, is a labeled graph built for a set of beliefs B. The labels map vertices to boolean functions of states. The structure is equivalent to a particular PG_{LUG}. Let $b^* = \bigcup_{b \in B} b$. Then $SLUG(B) = PG_{LUG}(b^*)$.*

Any query concerning $PG_{LUG}(b) = (V_b, E_b, l_b)$, for $b \in B$, can be answered using $SLUG(B)$ and the following true statements:

1. *$v \in V_b$ iff: $v \in V$ and $l(v) \wedge b$ is satisfiable.*
2. *$e = uv \in E_b$ iff: $e \in E$ and $l(u) \wedge l(v) \wedge b$ is satisfiable*
3. *$l_b(v) = l(v) \wedge b$*

By analogy with $h^{SA_R}_{lev}$, we define h^{SLUG}_{lev}. Let G denote Figure 2. This figure depicts the $SLUG$ built for the set of all beliefs. Let us consider determining $h^{PG_{LUG}}_{lev}(b,\mathcal{G})$ for the following belief, b: we are certain that our state contains exactly one vowel, and either the digit 0 or the digit 5. Here, $\mathcal{G}$ is the recurring goal of being certain of "b" and "5". For each level i, we can check $b \models G.l((f_b=\top,i))$ and $b \models G.l((f_5=\top,i))$ until both statements are true. This happens for the first time at level 5, since before that point states such as "a0" have not yet reached $f_5 = \top$. h^{SLUG}_{lev} formally captures this method, allowing us to share graphs across search states in belief space planning.

Definition 9 (h^{SLUG}_{lev}) *The function $h^{SLUG}_{lev}(G, b, \mathcal{G})$ is a level heuristic for belief-space planning. $G = SLUG(B)$ is built, once, for a set of belief-states, B, and re-used to derive a heuristic estimate for any $b \in B$:*

$$h^{SLUG}_{lev}(G, b, \mathcal{G}) = argmin_i \quad \begin{array}{c} \forall x, (\mathcal{G} \models x) \Rightarrow \\ ((x, i) \in G.V \wedge \\ b \models G.l((x, i))) \end{array}$$

Utilizing SAG

We utilize the SAG structure by applying it to progression planning. We build a SAG for many search states at once, and extract heuristics for each from the shared graph. The choice of scope has a great impact on the performance of any approach based on SAG. Using fewer states in the scope almost always requires less computation per graph. While not every label function becomes smaller, the aggregate size almost always decreases. Restricting the scope, however, prevents the SAG from representing the PGs for states so excluded. If such states are visited in search, then a new SAG will need to be generated to cover them. All of the approaches based on SAG can be seen as a particular strategy for covering the set of all states with shared graphs. We define 4 isolated points in that spectrum: Global-SAG, Reachable-SAG, Child-SAG, and the PG (Node-SAG). With respect to representing labels as Binary Decision Diagrams (BDDs) (Meinel & Theobald 1998) (as our implementation does), we provide an intuition concerning performance.

Global-SAG: A Global-SAG is a single graph for an entire planning episode. The scope is taken to be the set of all states; that is, Global-SAG uses the degenerate partition containing only the set of all states. Under a natural variable ordering of “a-z0-9”, we can consider the size of the label functions in our example domain. Each individual label function is a relatively small BDD; however, the pertinent statistic (since diagrams are shared) is that the total size of all the diagrams combined is 108.

Reachable-SAG: A Reachable-SAG is also a single graph for an entire planning episode. A normal planning graph is constructed from the initial state of the problem; all states consistent with the last level form the set from which the Reachable-SAG is built. That is, states are partitioned into two groups: definitely unreachable, and possibly reachable. A graph is generated only for the latter set of states.

With respect to the running example, consider the initial state of “5”. The last level of the planning graph built for that state allows all of the false literals, but among true literals, admits only “5-9”. So the set of states consistent with $\neg f_a \wedge \cdots \wedge \neg f_z \wedge \neg f_0 \wedge \cdots \wedge \neg f_4$ is what the Reachable-SAG is built for. Compared to the Global-SAG, the Reachable-SAG levels off after 5 levels instead of 10, and consume 52 shared BDD nodes (under the order “5-9a-z0-4”) instead of 108.

Child-SAG: A Child-SAG is a graph built for the set of children of the current node. This results in one graph per non-leaf search node. That is, Child-SAG partitions states into sets of siblings. While this is still an exponential number of graphs, the reduction, relative to Node-SAG, is still significant.

Node-SAG: A Node-SAG is built for the smallest conceivable scope: the current search node. In such a situation, label propagation is useless, and therefore skipped. That is, Node-SAG is just a name for the naive approach of building a PG for every search node.

Empirical Evaluation

In order to qualitatively evaluate our techniques, we developed a belief-space progression planner called POND. We implemented the SAG strategies (Node,Child,Reachable,Global) within POND. The degenerate case, Node-SAG, simply uses PG_{LUG}. The other three cases are implemented with respect to the (optimized) state agnostic version of PG_{LUG}: $SLUG$.

We discuss the implementation, and in particular, the relaxed plan based heuristic employed throughout our experiments. We report on our internal comparison of SAG-based strategies. We demonstrate that POND (using the best strategy from the internal comparison) is competitive with state of the art belief space planners. Domains, problems, POND, and the full set of test results are all available at *http://rakaposhi.eas.asu.edu/belief-search/*.

Implementation: POND searches in the space of belief states to find strong conditional plans (conformant and classical plans are special cases). Searching for conditional plans requires more complicated methods than A* and its cousins; we use AO* (And/Or) search (Nilsson 1980) to find conditional plans. POND is implemented in C++ and makes use of many software packages. The search relies on the LAO* source code (Hansen & Zilberstein 2001) (modified to only do AO*). As previously mentioned, labels are represented as BDDs (Meinel & Theobald 1998). Likewise, we represent actions and belief states as BDDs, via the CUDD library (Brace, Rudell, & Bryant 1990). We progress actions over beliefs using the BDD image function, as in MBP (Bertoli *et al.* 2001). As previously implied, POND extends the planning graph implementation provided in the IPP planner (Koehler 1999); we include support for propagating our label functions and sharing graphs across multiple search states.

In all of experiments, we use a belief-space relaxed plan heuristic (Bryce & Kambhampati 2004) to guide POND’s AO* search. We additionally weight the heuristic: $f = g + 5 * h$. Our relaxation ignores sensing in addition to ignoring negative interactions between operators. We extract a belief-space relaxed plan from $PG_{LUG}(b)$ by requiring that, for any $s \in b$, the set of vertices whose labels’ evaluate to true for s represents a classical relaxed plan. Prior work (Bryce, Kambhampati, & Smith 2004) demonstrates a procedure for efficiently extracting such a plan via label algebra (as opposed to enumerating the states in the belief). The relaxed plan can be viewed as ignoring negative interactions between states in a belief in addition to ignoring negative interactions between operators. We extend this to extracting belief-space relaxed plans from $SLUG(B)$, for $b \in B$, by conjoining the labels of the $SLUG$ with b before using them

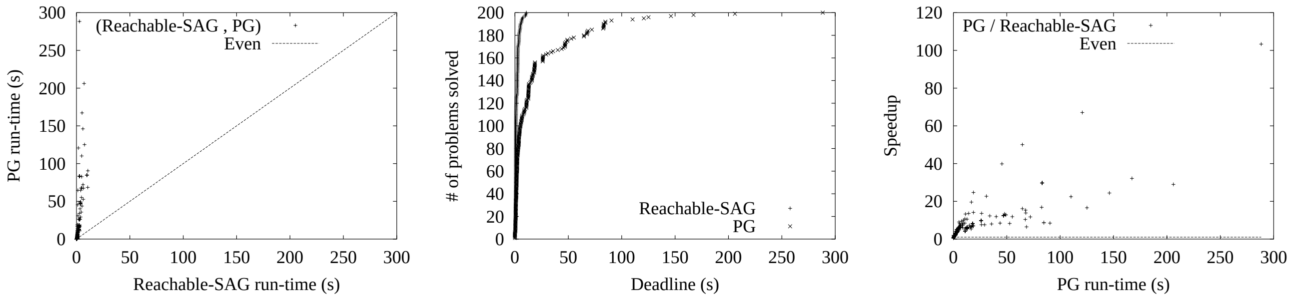

Figure 3: Reachable-SAG (using $SLUG$) vs. PG (using PG_{LUG}), Belief-Space Problems

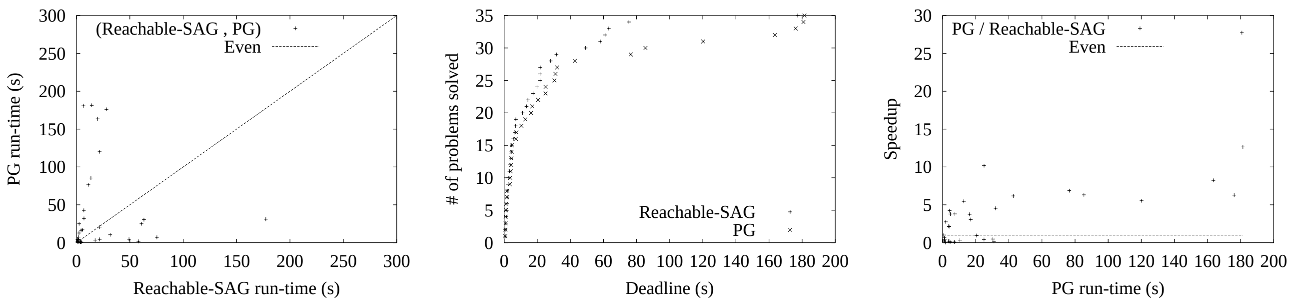

Figure 4: Reachable-SAG (using $SLUG$) vs. PG (using PG_{LUG}), Classical Problems

in the above extraction procedure. As with classical relaxed plans, the heuristic value of a belief-space relaxed plan is the number of action vertices it contains.

Internal Comparison

We ran tests across a wide variety of benchmark problems in belief-space planning. All problems in the POND distribution were attempted; we further augmented the test set with all benchmarks from the 2000, 2002, and 2004 planning competitions. The latter problems are classical planning problems, which we include in order to gauge the possible benefits of applying SAG to classical planners.

As there are several hundred problems tested, we imposed relatively tight limits on the execution (5 minutes on a P4 at 3.06 GHz with 900 MB of RAM) of any single problem. We exclude failures due to these limits from the figures. In addition, we sparsely sampled these failures with relaxed limits to ensure that our conclusions were not overly sensitive to the choice of limits. Up until the point where physical memory is exhausted, the trends remain the same.

Our measurement for each problem is the total run time of the planner, from invocation to exit. We have only modified the manner in which the heuristic is computed; despite this, we report total time to motivate the importance of optimizing heuristic computation. It should be clear, given that we achieve large factors of improvement, that time spent calculating heuristics is dominating time spent searching.

Figures 3 and 4 provide several perspectives on the merits of Reachable-SAG. The figures omit Global-SAG and Child-SAG for clarity. The former, Global-SAG, is dominated; the mode wastes significant time projecting reachability for unreachable states. The latter, Child-SAG, improves upon the PG approach in virtually all problems. However, that margin is relatively small, so we prefer to depict the current standard in the literature, the PG approach. The first graph in each figure is a scatter-plot of the total running times. The line "y=x" is plotted, which plots identical performance. The second graph in each figure plots the number of problems that each approach has solved by a given deadline. The third graph in each figure offers one final perspective, plotting the ratio of the total running times.

Belief-Space Domains The scatter-plots reveal that Reachable-SAG always outperforms the PG approach. Moreover, the boost in performance is well-removed from the break-even point. The deadline graphs are similar in purpose to plotting time as a function of complexity: rotating the axes reveals the telltale exponential trend. However, it is difficult to measure complexity across domains. This method corrects for that at the cost of losing the ability to compare performance on the same problem. We observe that, with respect to any deadline, Reachable-SAG solves a much greater number of planning problems. Most importantly, Reachable-SAG out-scales the PG approach. When we examine the speedup graphs, we see that the savings grow larger as the problems become more difficult.

Classical Domains POND treats classical problems as problems in belief-space; naturally, this is very inefficient. Despite this, Reachable-SAG still produces an improvement on average. While the scatter-plots reveal that performance can degrade, it is still the case that average time is improved: mostly due to the fact that as problems become more diffi-

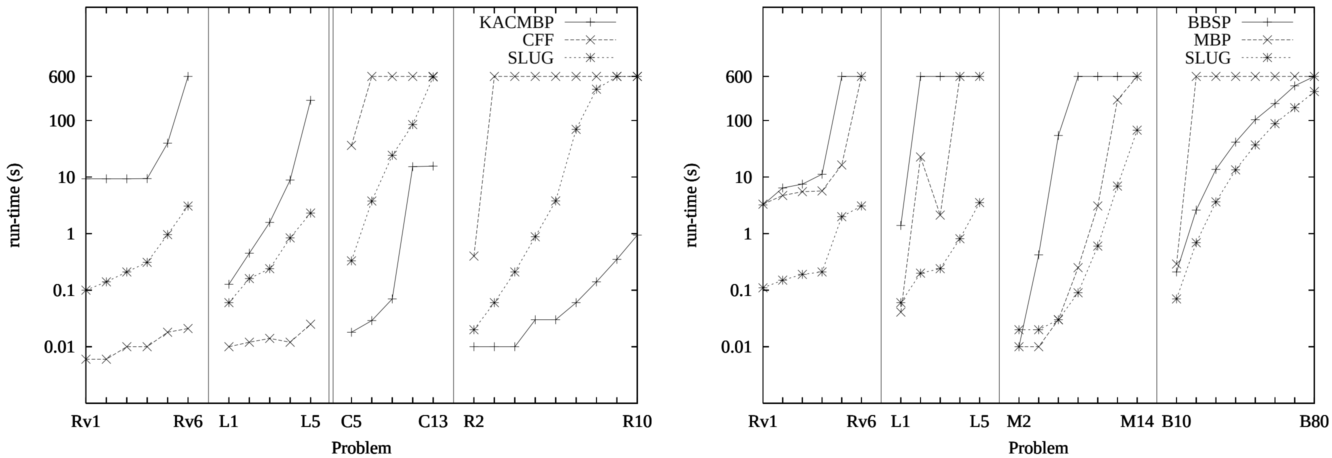

Figure 5: Comparison of planners on conformant (left) and conditional (right) domains. Four domains appear in each plot. The conformant domains are Rovers(Rv1-Rv6), Logistics(L1-L5), Cube Center(C5-C13), and Ring(R2-R10). The conditional domains are Rovers(Rv1-Rv6), Logistics(L1-L5), Medical(M2-M14), and BTCS(B10-B80).

cult, the savings become larger.

In light of this, we have made a preliminary investigation in comparing SA_R to state of the art implementations of PG_R. In particular, Hoffmann & Nebel (2001) go to great lengths to build PG_R as quickly as possible, and subsequently extract a relaxed plan. We attempted to compete against that implementation with a straightforward implementation of SA_R. We ran trials of greedy best-first search using $h_{RP}^{PG_R}$ (the relaxed plan heuristic) against using $h_{RP}^{SA_R}$ with the Reachable-SAG strategy. Total performance was improved, sometimes doubled, for the Rovers domain; however, in most other benchmark problems, the relative closeness of the goal and the poor estimate of reachability prohibited any improvement. Of course, per-node heuristic extraction time (i.e. ignoring the time it takes to build the shared graph) was always improved, which motivates an investigation into more sophisticated graph-building strategies than Reachable-SAG.

External Comparison

We made an external comparison of our planner POND with several of the best conformant: KACMBP (Bertoli & Cimatti 2002) and CFF (Brafman & Hoffmann 2004), and conditional planners: MBP (Bertoli *et al.* 2001) and BBSP (Rintanen 2005). Based on the results of the internal analysis, we used belief-space relaxed plans extracted from a common $SLUG$, using the Reachable-SAG strategy. We denote this mode of POND as "SLUG" in Figure 5. The tests depicted in Figure 5 were allowed 10 minutes on a P4 at 2.8 GHz with 1GB of memory. The planners we used for these comparisons require descriptions in differing languages. We ensured that each encoding had an identical state space; this required us to use only boolean fluents in our encodings.

Conformant Domains We used the conformant Rovers and Logistics domains (Bryce & Kambhampati 2004) as well as the Cube Center and Ring domains (Bertoli & Cimatti 2002) for our conformant planner comparison in Figure 5. These domains exhibit two distinct dimensions of difficulty. The primary difficulty in Rovers and Logistics problems centers around *causing* the goal. The Cube Center and Ring domains, on the other hand, revolve around *knowing* the goal. The distinction is made clearer if we consider the presence of an oracle. The former pair, given complete information, remains difficult. The latter pair, given complete information, becomes trivial, relatively speaking.

We see our heuristic as a middle-ground between KACMBP's cardinality based heuristic and CFF's approximate relaxed plan heuristic. In the Logistics and Rovers domains, CFF dominates, while KACMBP becomes lost. The situation reverses in Cube Center and Ring: KACMBP easily discovers solutions, while CFF wanders. Meanwhile, by avoiding approximation and eschewing cardinality in favor of reachability, POND achieves middle-ground performance on all of the problems.

Conditional Domains We devised conditional versions of Logistics and Rovers domains by introducing sensory actions. We also drew conditional domains from the literature: BTCS (Weld, Anderson, & Smith 1998) and a variant of Medical (Petrick & Bacchus 2002). Our variant of Medical splits the multi-valued stain type sensor into several boolean sensors.

The results (Figure 5) show that POND dominates the other contingent planners. This is not surprising: MBP's heuristic is belief state cardinality, and BBSP uses no heuristic. Meanwhile, POND employs a strong, yet cheap, estimate of reachability (relaxed plans extracted from $SLUG$, in Reachable-SAG mode). MBP employs greedy depth-first search, so the quality of plans returned can be drastically poor. The best example of this in our results is instance Rv4 of Rovers, where the max length branch of MBP requires 146 actions compared to 10 actions for POND.

Related Work

We have already noted that our work is a generalization of (Bryce, Kambhampati, & Smith 2004), which efficiently exploits the overlap in the PGs of members of a belief. Liu, Koenig, & Furcy (2002) have explored issues in speeding up heuristic calculation in HSP. Their approach utilizes the prior PG to improve the performance of building the current PG (the rules which express the dynamic program of HSP's heuristic correspond to the structure of a PG). We set out to perform work ahead of time in order to save computation later; their approach demonstrates how to boost performance by skipping re-initialization. Also in that vein, Long & Fox (1999) demonstrate techniques for representing a PG that take full advantage of the properties of the PG. We seek to exploit the overlap between different graphs, not different levels. Liu, Koenig, & Furcy seek to exploit the overlap between different graphs as well, but limit the scope to graphs adjacent in time. Another way of avoiding the inefficiencies in repeated construction of PGs is to do the reachability computation in the backward direction (Kambhampati, Parker, & Lambrecht 1997). However, we note that state of the art progression planners typically do reachability analysis in the forward direction. Work on greedy regression graphs (McDermott 1999) as well as the GRT system (Refanidis & Vlahavas 2001), can be understood this way.

Conclusion

A common task in many planners is to compute a set of planning graphs. The naive approach fails to take account of the redundant sub-structure of planning graphs. We developed the state agnostic graph (SAG) as an extension of prior work on the labeled uncertainty graph (PG_{LUG}). The SAG employs a labeling technique which exploits the redundant sub-structure, if any, of arbitrary sets of PGs.

We developed a belief-space progression planner called POND to evaluate our technique. We improve the use of the PG_{LUG} within POND by applying our SAG technique. We found an optimized form, $SLUG$, of the state agnostic version of the PG_{LUG}. This optimized form improves worst-case complexity, which carries through to our experimental results.

We compared POND to state of the art planners in conformant and conditional planning. We demonstrated that, by using $SLUG$, POND is highly competitive with the state of the art in belief-space planning. Given our positive results in applying SAG, we see promise in applying SAG to other planning formalisms.

Acknowledgements: This research is supported in part by the NSF grant IIS-0308139 and an IBM Faculty Award to Subbarao Kambhampati. We thank David Smith for his contributions to the foundations of our work, in addition, we thank the members of Yochan and Subbarao Kambhampati for many helpful suggestions.

References

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *AIPS*, 143–152.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, 473–486.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *IJCAI*, 1636–1642.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 360–372.

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

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *ICAPS*, 355–364.

Bryce, D., and Kambhampati, S. 2004. Heuristic guidance measures for conformant planning. In *ICAPS*, 365–375.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2004. Planning in belief space with a labelled uncertainty graph. In *AAAI Workshop on Learning and Planning in Markov Decision Processes.*

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *JAIR* 20:239–290.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kambhampati, S.; Parker, E.; and Lambrecht, E. 1997. Understanding and extending graphplan. In *ECP*, 260–272.

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

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding up the calculation of heuristics for heuristic search-based planning. In *AAAI/IAAI*, 484–491.

Long, D., and Fox, M. 1999. Efficient implementation of the plan graph in stan. *JAIR* 10:87–115.

McDermott, D. V. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence* 109(1-2):111–159.

Meinel, C., and Theobald, T. 1998. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer.

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

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann.

Petrick, R. P., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS*, 212–221.

Refanidis, I., and Vlahavas, I. 2001. The GRT planning system: Backward heuristic construction in forward state-space planning. *JAIR* 15:115–161.

Rintanen, J. 2003. Expressive equivalence of formalisms for planning with sensing. In *ICAPS*, 185–194.

Rintanen, J. 2005. Conditional planning in the discrete belief space. In *IJCAI*.

Weld, D. S.; Anderson, C.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *National Conference on Artificial Intelligence*. AAAI Press.

Younes, H., and Simmons, R. 2003. Vhpop: Versatile heuristic partial order planner. *JAIR* 20:405–430.

Zimmerman, T., and Kambhampati, S. 2005. Using memory to transform search on the planning graph. *JAIR* 23:533–585.