The Footprint Principle for Heuristics for Probabilistic Planners

Jim Blythe
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
jblythe@cs.cmu.edu

Abstract. Probabilistic back-chaining planners, which use probabilities to represent and reason about uncertainty in the planning domain, typically have a larger search space than their classical counterparts. Therefore heuristics that can reduce their search effectively are even more important. The "footprint" principle leads to a family of heuristics for probabilistic planners produced by attempting to make subsequent refinements to a plan apply to disjoint sets of planning cases. Heuristics derived by this principle are shown to be effective for two probabilistic planners, Buridan and Weaver, which are organised around quite different search techniques. Probabilistic planners are needed that can use more compact representations of uncertainty than those that currently exist, and these planners will depend even more on the footprint principle and others like it.

1 Introduction

Probabilistic planners that use classical back-chaining algorithms, such as [6, 1, 4, 7], extend the range of problems addressable by classical AI planners to include uncertainty, which might be present in the initial state, in the outcomes of actions and in external events in the domain, using representations based on probability theory. Rather than seek a plan that will definitely transform an initial state to one matching a goal description, they seek a plan that will achieve the goal description with probability greater than some threshold value, given a probability distribution over initial states.

Search control heuristics are perhaps even more important for probabilistic planners than for classical planners. This is because the search space for probabilistic planners is at least as large as for classical planners, which they include as a special case. Probabilistic planners often need to consider different initial states, either implicitly or explicitly, and branching timelines due to the multiple possible effects of actions. These planners may also frequently compute the probability of success of candidate plans, a global operation that can be expensive.

In this paper I outline a strategy, called the footprint principle, for deriving effective search heuristics for a wide range of probabilistic planners. The strategy creates heuristics that, given a plan, seek refinements to it that cover new sets of planning cases as distinct as possible from those already covered, in a sense made precise in the next section. The strategy helps to maximise the benefit of the refinement in terms of the increase in the plan's probability of success.

The next section provides a formal description of planning under uncertainty that is used to motivate and explain the footprint principle, and the related coherence principle. The following two sections show examples of heuristics derived with the footprint principle for two different probabilistic planners, Buridan [6] and Weaver [1]. Finally I argue that planners should be based on more compact representations of uncertainty than either Buridan or Weaver are currently, one which will more naturally deal with sets of states. It is likely that such planners will make more use of the coherence and footprint principles than do present planners.

2 Uncertain planning and the footprint principle

I begin with a formal description of planning under uncertainty, similar to [6] that is used to discuss the coherence and footprint heuristics and describe the different algorithms.

2.1 Description of planning under uncertainty

A planning domain is defined by a state space, Ω, and a set of actions $\mathcal{A}$ that map a state to a probability distribution over Ω. For AI planners the actions typically are defined using a set of literals L, each member of which denotes a subset of Ω. For example, let $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ and let $L = \{$ engine-ok, car-starts $\}$, where engine-ok $= \{\omega_1, \omega_2\}$ and car-starts = $\{\omega_2, \omega_3\}$. I define the set of literals that are true for a state $\omega \in \Omega$ (that is, those literals containing ω) as lit(ω). Expressions can be formed using the literals and the logical connectives $\wedge$, $\vee$ and $\neg$ to denote sets of states. For example "engine-ok $\wedge \neg$ car-starts" denotes the singleton set $\{\omega_1\}$. Defining literals in terms of states, rather than the other way around, allows us to conveniently represent redundant literal sets, insufficiently descriptive literal sets and inconsistent expressions. For the purposes of this paper, however, the reader may assume that there is a unique state $\omega \in \Omega$ corresponding to each full assignment of truth values to the literals L. Thus, a state $\omega \in \Omega$ is uniquely defined by lit(ω).

Actions are defined in terms of their effects on literals rather than directly on states, as an extension to the action definition in STRIPS. An action a in STRIPS, which includes no representation of uncertainty, is specified by a precondition written pre(a), which is a logical expression over the literals, and two lists of literals called the add and delete lists, written add(a) and del(a) respectively. If the action is performed in the state ω, in which the precondition is satisfied, the resulting state ω' is specified by its literal set:

$$\text{lit}(\omega') = (\text{lit}(\omega) - \text{del}(a)) \cup \text{add}(a)$$

This state is written as result$(\omega, \text{add}(a), \text{del}(a))$.

An action in a probabilistic planner maps a state in Ω to a probability distribution over Ω. First the effects are generalised to a finite probability distribution over add and delete lists, called an *effect distribution*. This is a list of triples, $(p_i, \text{add}_i, \text{del}_i)$, with $1 \leq i \leq n$ for some finite n, where the add$_i$ and del$_i$ are add and delete lists of literals as above, $p_i > 0$ for $1 \leq i \leq n$ and $\sum_{i=1}^{n} p_i = 1$. An action a is specified by a binary tree whose leaves correspond to effect distributions and whose internal nodes correspond to literals. When the action is applied in a state ω, the tree is first traversed to find the appropriate effect distribution. At each internal node of the tree, the "true" branch is taken if the associated literal is in lit(ω), and otherwise the "false" branch is taken. The effect distribution at the end of this path gives a conditional probability distribution $P(.|\omega, a)$ over Ω defined by

$$P(\omega'|\omega, a) = p_i, \text{ if } \omega \in \text{pre}(a) \text{ and lit}(\omega') = \text{result}(\omega, \text{add}_i, \text{del}_i) \text{ for some } i$$

$$P(\omega'|\omega, a) = 0 \text{ otherwise}$$

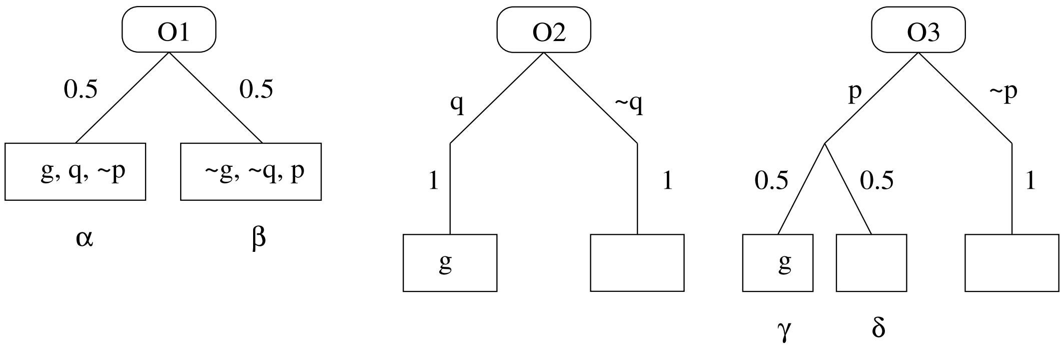


Figure 1: The operators for an example domain.

A planning problem is specified by an initial probability distribution P_I over Ω, a goal expression built from literals in L and a threshold probability p_t. A solution to a planning problem is a partially ordered set of actions, which when executed in any linear extension of the partial order will achieve a state matching the goal expression with probability at least p_t, given that the initial state comes from the distribution P_I. Given the conditional distribution defined above, we can write down the probability that some expression E is true after performing the action a, given the initial state distribution P_I:

$$\sum_{\omega \in \Omega} \sum_{i \in \Omega} P(E|\omega) P(\omega|i, a) P_I(i)$$

Note that $P(E|\omega)$ is 1 if E is true in state ω, and 0 otherwise. The probability $P(E|a)$ can therefore be simplified by summing over $E \subseteq \Omega$ to

$$\sum_{\omega \in E} \sum_{i \in \Omega} P(\omega|i, a) P_I(i)$$

I write this as $P(E|a)$, leaving P_I implicit.

Plans are represented according to the following rules. The empty plan [] is a plan, and so is any action a. If π_1 and π_2 are plans then so is the sequence π_1 followed by π_2, written $\pi_1; \pi_2$, and if E is an expression then the branching plan "If E then π_1 otherwise π_2" is also a plan. This is written $[E \ ? \ \pi_1 \mid \pi_2]$. I extend the syntax to write $P(E|a; b)$ for the probability that E is true after performing a followed by b, which can be written as a triple sum in the same was as $P(E|a)$ above.

Given a plan π, I define a *trace* of the plan to be a tuple $(\omega, \text{out}_1, \text{out}_2, \ldots)$ consisting of an initial state $\omega \in \Omega$ and outcomes for each of the actions taken in the plan. In the absence of external events, every possible outcome when a plan is executed in an initial state is represented by a trace[1]. The set of all traces for a plan π and an initial state ω is written tr(π, ω) and the (disjoint) union of all traces for all possible initial states is written tr(π). The set of traces of a plan is not needed for a formal description of planning under uncertainty, but it is useful for explaining the footprint principle.

As an example, consider a domain described by three literals, $L = \{g, p, q\}$ with initial state $\omega_I = (\neg g, \neg p, \neg q)$. The three operators are illustrated in Figure 1. Operator O_1 is specified by a single effect distribution, which with probability 0.5 adds g and q and deletes p (outcome α), and with probability 0.5 deletes g and q and adds p (outcome β). Operator O_2 adds g with probability 1 if q is true, and otherwise has no effect. If p is true, Operator O_3 adds g with probability 0.5 (outcome γ) and has no effect with probability 0.5 (outcome

[1]In the terms of Markov decision processes as described in [2], a trace is a trajectory in the MDP corresponding to the problem.

δ). If p is false, it has no effect. Given a plan $\mathcal{P} = [O_1]$, the set of traces with respect to ω_I is $\text{tr}(\mathcal{P}, \omega_I) = \{(\omega_I, O_1 : \alpha)(\omega_I, O_1 : \beta)\}$.

Note that the traces are always mutually exclusive and exhaustive, and each one can be assigned a probability derived from the initial state distribution and the probability of each outcome, so that the probability of some property holding after the plan is executed is found by summing the probabilities of the traces in which the property holds. In this example, each trace has probability $1/2$, and the probability that g is true after executing the plan is also $1/2$, since it is true only in the first trace.

2.2 *The footprint principle for probabilistic planners*

Probabilistic planners build plans by incrementally refining some candidate plan to increase its probability of success, stopping when the threshold probability is reached and backtracking if no extensions achieve the desired probability. This incremental refinement might for example be the addition of a new conditional branch of the plan, or some extra steps to improve the probability of some subgoal in the plan. The heuristics based on the footprint principle aim to increase the rate of convergence to a good plan, by choosing improvements to an existing plan that are more likely to significantly increase the probability of success. This is similar to the maximum expected utility approach of Russel and Wefald [9]. While the exact heuristics may differ between different planners, the underlying principles are unchanged, and they are explained in this section. I begin with two observations about the process of improving the probability of success of plans in uncertain domains.

Firstly, while plans in STRIPS-style domains are created for individual initial states, plans in probabilistic domains are created for a set of states, those with sufficiently high probability in the initial distribution to influence the probability of plan success. I define a *coherent* set of traces for a given plan with respect to a plan refinement to be a set for which this single refinement can lead to a goal state with high probability. It can be a valuable strategy for a probabilistic planner to expend computation looking for a coherent set of traces with a high probability mass, and this I refer to as the "coherence" strategy.

The second observation is that the increase in the probability of success due to a plan refinement is not additive, since the new refinement to the plan typically does not work in a completely different set of cases from the original. Recall that if E and F are expressions, $P(E \vee F) = P(E) + P(F) - P(E \wedge F)$. Thus an estimate of $P(E \vee F)$ found by summing $P(E)$ and $P(F)$ overestimates to the extent that E and F overlap. This can happen when estimates of the probability of success of the refined plan are made from the probability of the original plan and an estimate of the goodness of the refinement.

To illustrate this, consider the example scenario of Section 2.1 in which the planner has the goal g and a partial plan consisting of the operator O_1. The task is to choose an operator to add to the plan to improve its probability of success.

A comparison of the operators O_2 and O_3 that assumes independence might lead the planner to prefer O_2, since p and q are both true with probability $1/2$ after O_1 is applied, so O_2 adds g with probability $1/2$ but O_3 adds it with probability only $1/4$. However the plan $[O_1 ; O_2]$ does not improve over $[O_1]$ since p is true only when O_1 achieves g. The plan $[O_1 ; O_3]$ has probability $3/4$ of success because O_3 precisely addresses the case when O_1 fails. In the general case, comparisons of operators based on their "average" ability to achieve goals tell us nothing about their usefulness for refining plans.

This simple observation about independence becomes important when we consider heuristics that estimate the increase in probability of success provided by refining a plan. I define the *footprint* of a plan as the set of traces in which it succeeds. Note that when we refine a plan by adding a step, each trace of the new plan is an extension of some trace in the old plan, so the traces in the footprint of the new plan can be classified into two categories: those that extend traces in the footprint of the original plan and those that extend other traces. The new traces that extend traces in the original footprint form the intersection of the refinement's footprint with that of the original plan, and do not improve the probability of success.

Heuristics that estimate the goodness of the refinement without regard to the existing plan, such as noting the maximum or average probability that some step will add a desired subgoal, are likely to over-estimate the efficacy of the refinement when its footprint has a significant intersection with the footprint of the original plan, in terms of its probability mass. This was the case for O_2 — the footprint of $[O_1; O_2]$ is completely contained within the footprint of $[O_1]$. If instead relatively cheap heuristics can be designed that take some of the overlap into account, they can be considerably more powerful than those that assume independence. I describe heuristics built on this principle as belonging to the "footprint" family. Combining the two observations in this section, by discounting the overlap between old and new parts of the plan these heuristics can be viewed as seeking refinements to the plan that have a footprint with high probability mass that is disjoint from the footprint of the original plan. Alternatively, these refinements can be viewed as having a high-probability footprint, conditional on the original plan failing.

3 Case study: Buridan

3.1 Buridan's algorithm and a worked example

Buridan [6] is based on the causal-link planner SNLP [8], and searches the space of possible plans. Actions in Buridan follow the representation described in section 2.1, with a tree of effect distributions. The conjunction of literals on a path through the tree is called a "trigger" and each element of the associated effect distribution, taken with the trigger, is a set of "consequences", one for each element of the effect distribution. Buridan differs from SNLP in its use of causal links and its representation of subgoals in plans. Buridan's causal links include a reference to a particular consequence of an action, and each plan includes a list of subgoals, $p@a_i$, for each atom p in each trigger used in a causal link, where an atom is either a literal or negated literal. Whereas SNLP has at most one causal link for each precondition in a plan, Buridan may have many.

Buridan begins with a plan with two actions, initial and goal, which set up the initial and goal conditions. Buridan halts when it selects a plan whose probability of success is greater than the threshold. Otherwise it tries to improve the probability of success of the current plan by one of the following refinement steps:

1. Add a link to the subgoal $p@a_i$, choosing from all actions, whether in the current plan or not, that have a consequence that can achieve p. The linked action is ordered before a_i in the partial order.

2. Specialise the partial order to resolve a threat, as in SNLP

3. "Confront" a threat by planning to reduce the probability of the threatening effect taking place.

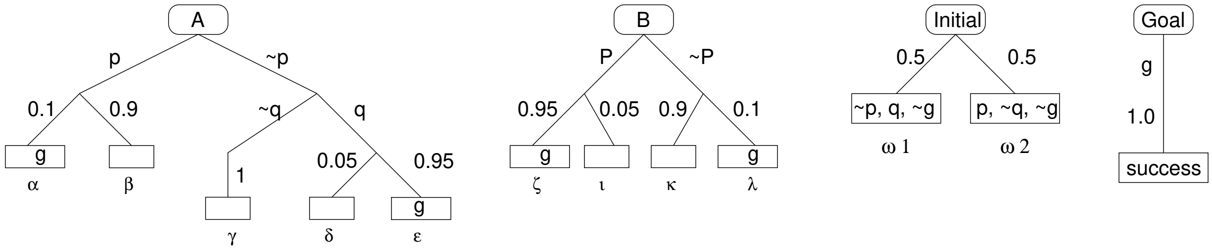


Figure 2: Example domain 1

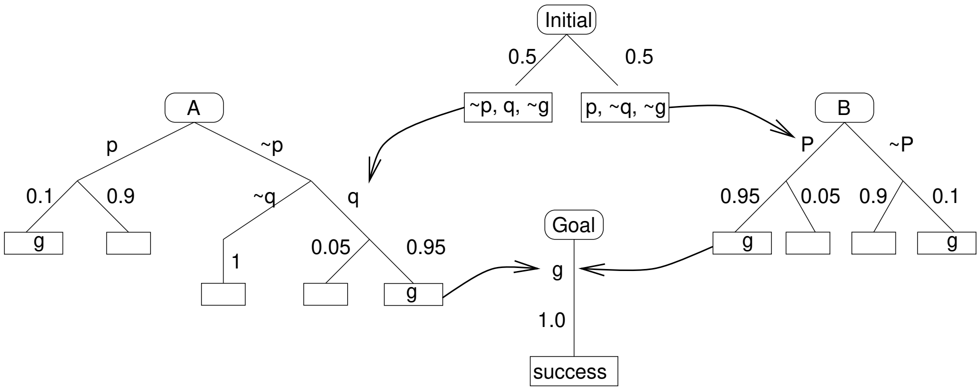


Figure 3: A solution in Buridan to example problem 1

More details about Buridan's algorithm can be found in [6].

The following example shows how Buridan works and is also used to describe a heuristic for Buridan's planner based on the footprint principle. The actions are defined as shown in Figure 2, with n identical copies of the action B, where n is a parameter to the problem. For example, A has three triggers, p, $\neg\mathsf{p} \wedge \mathsf{q}$ and $\neg\mathsf{p} \wedge \neg\mathsf{q}$. The effect distribution for trigger p is $\{[0.1\ (\mathsf{g})\ ()]\ [0.9\ ()\ ()]\}$. Thus, if A is applied in a state where p is true, it adds g with probability 0.1 and has no effect with probability 0.9. The initial distribution consists of two equally likely states and the goal is the literal g, to be achieved with probability at least 0.9.

We assume that Buridan's non-deterministic algorithm makes all the right choices and leads directly to a plan for this example. Buridan, then, adds a link to g from one of the possible actions, say A, choosing the trigger $\neg\mathsf{p} \wedge \mathsf{q}$. Then a link is added from the first outcome of the initial state distribution. This yields a plan that achieves g with probability 0.525, although Buridan by default calculates a lower bound on the probability by only accounting for the explicit links in a plan, and sees a lower bound of 0.475. Buridan then adds a second link to g, choosing action B and trigger p, and links this to the second outcome of the initial state distribution. Following the explicit links of the plan, a lower-bound probability of success of 0.95 is calculated. The true probability of success of either linearisation of this plan is higher due to the effects of the triggers not mentioned in the causal links of the plan. Buridan returns this plan, shown in Figure 3 with causal links represented by the bold lines.

3.2 Search control and heuristics in Buridan

We have seen Buridan's top-level loop and the plan refinement operations available that determine its search space, without covering how the search is organised. Buridan performs a best-first search, at each stage computing all possible modifications of the current plan, inserting them into a list of untried plans ordered by a "ranking function" and repeatedly

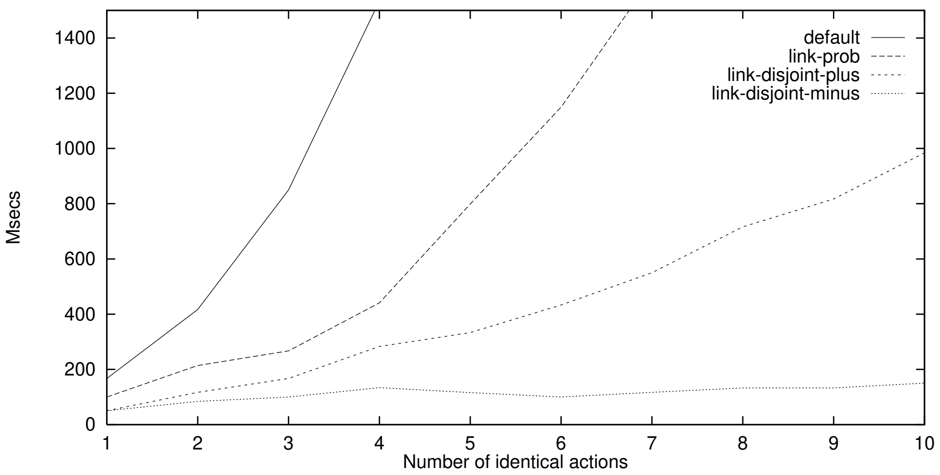


Figure 4: Timings for different ranking functions on the example problem.

selecting the first element of this list. By default, the ranking function for a plan seeks to minimise the sum of the steps, threats, unlinked triggers and causal links in the plan, essentially leading to a breadth-first search. While both the search method and ranking function can be changed by the user, in the experiments below the search method was unchanged and different ranking functions were used.

Four different ranking functions were tried, default, link-prob, link-disjoint-plus and link-disjoint-minus, which differ only in the way they treat causal links. The first, default, is the default ranking function described above, which performed the best of the ranking functions supplied with Buridan. The second, link-prob replaces the number of links in the expression to be minimised with the sum $\sum_{l \in \mathcal{L}} 1 - p_l$, where p_l is the probability associated with the consequence as a member of an effect probability distribution, and $\mathcal{L}$ is the set of links in the plan. Thus link-prob takes the strengths of the consequences into account, but not the extent to which they may be independent.

The third and fourth ranking functions compute the same function of the links in the plan, but use it in different ways. I will describe how they differ below, but first I motivate and describe the function of the links, which is an approximation of the footprint principle. Consider the example of section 3.1. The one-step plan $[A]$ is improved by adding a second step to achieve the same goal condition g. If the second step is chosen without reference to the current plan, both A repeated and B look equally good, with links that add g with probability 0.95. However that measurement ignores the potential overlap between the traces covered by the refinements and the original plan. The refinement $[A; A]$ has a larger overlap with the footprint of the original plan, and therefore $[A; B]$ yields a larger increase in the probability of success.

I apply a function sep to the links that penalises multiple links supporting the same trigger, unless it can show that they apply to disjoint sets of traces by analysing their trigger conditions. For each condition c in the plan, let $l(c)$ be the set of causal links in $\mathcal{L}$ that establish the condition c. These represent multiple steps for the same goal. We look for a literal m such that each link in $l(c)$ contains either m or $\neg m$ in its trigger, and there is at least one link with m and one with $\neg m$ in its trigger. If such an m is found the links are split into two sets, those with m and those with $\neg m$ in the trigger, otherwise the links are put into one set. When all the links are partitioned into subsets s_i, we compute $\mathsf{sep}(\mathcal{L}) = \sum_{s_i} \max_{l \in s_i} p(l)$, where $p(l)$ is the probability associated with the link l. (This value is also a function of the chosen literals m.)

sep is a measure of the support given to the trigger by its links that attempts to be pessimistic about links that potentially overlap, by taking the maximum of their probabilities (corresponding to the largest possible overlap in their footprints). It can be computed in linear time in the number of causal links. The ranking function link-disjoint-plus is then defined by replacing the term for causal links in link-default by $|\mathcal{L}| - \mathsf{sep}(\mathcal{L})$, a number that is always positive but lower when the link probabilities are higher and when they are more disjoint. The rank link-disjoint-minus does not include the $|\mathcal{L}|$ term, so that adding a link never increases the ranking function.

For example consider the two refinements $[A; A]$ and $[A; B]$ of the original plan $[A]$ to achieve g. Each refinement has two links to the goal condition g. In the first refinement the links are identical with triggers $\neg \mathsf{p} \wedge \mathsf{q}$, so no atom that differentiates the triggers can be found, and the contribution to sep from g is 0.95. In the second refinement the triggers are $\neg \mathsf{p} \wedge \mathsf{q}$ and p, as in Figure 3, so p splits the links and the contribution to sep is 1.9.

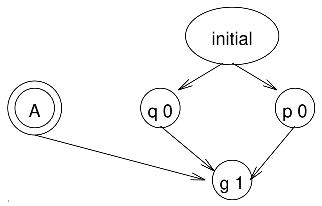
An experiment was made in which the number n of operators in the planning domain that were identical to B was varied. Real-world planning domains often contain many different operators that are almost identical, for example using different trucks to move a payload. Experiments varying the probability threshold and in other domains produced similar results. Figure 4 shows cpu times on a Sparc 20 for Buridan to solve the example problem of section 3.1 with the four heuristics as n varied. link-prob's naive use of probability measures gives a significant improvement over default, and link-disjoint-plus yields a similar improvement by taking the footprint into account. All three heuristics give roughly quadratic performance, which is to be expected since they will all lead Buridan to include plans with every combination of two steps. link-disjoint-minus does not have this behaviour because it eagerly seeks links, and has a roughly linear performance. Despite the dramatic difference between link-disjoint-minus and link-disjoint-plus, the footprint principle is at least as important a factor in the success of link-disjoint-minus, since the link-seeking variants of default and link-prob performed very poorly.

4 Case study: Weaver

Our second case study is with Weaver [1], a probabilistic planner based on Prodigy [3] and therefore a total-order planner. Weaver repeatedly calls Prodigy as a subroutine, choosing an initial state for Prodigy and a goal so that the result can be inserted into the emerging plan as a refinement. The domain for Prodigy is altered to include one operator for each individual outcome of a probabilistic operator. By default, Weaver performs a depth-first search of possible plan refinements. Weaver evaluates its plans using a Bayesian belief net that is built incrementally as the plan is refined. The nodes of the belief net are time-stamped and represent either steps in the plan or fluents at specific times. The true probabilistic nature of the actions is represented in the conditional probability distributions in the net. In this section, I demonstrate Weaver on the problem of the previous section and one of starting a potentially faulty car, and show how the footprint principle can find heuristics for this algorithm.

4.1 Weaver's algorithm and a worked example

Weaver can use parameterised operator schemas and can plan with uncertain external events represented in a similar model. This is beyond the scope of this paper but is described in [1]. A simplified version of Weaver's algorithm that ignores external events can be summarised



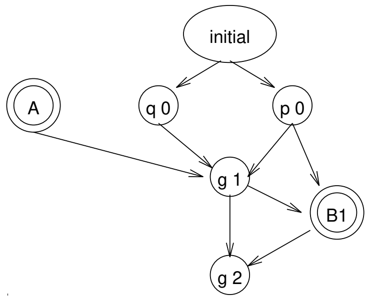


Figure 5: Belief net for the first stage plan and final plan to solve the first example problem. Double circles correspond to actions, the others to literals at different stages of the plan's execution.

as follows:

Initialise P to the empty plan, and initialise the belief net B to describe the initial state distribution.

Loop: If B satisfies the threshold goal probability, return P.
Otherwise find a set F of pairs (n, a) where n is a literal node in the belief net B with low probability of taking a desired value, and a is a step in the plan that may cause n to take an undesired value.
For each pair $f \in F$ try each of the following plan refinements:

- Call Prodigy to create a plan to achieve the goal state from a state caused by the action a, in which n does not take a desired value. Add the new plan segment as a branch in the plan, choosing some observable literal l that has different values from the desired outcome of a and from at least one undesired outcome as the test for the branch.

- Re-start Prodigy at the point where a was chosen, locally annotating its preconditions to forbid an undesired consequence.

If no refinement works, backtrack.
Otherwise, update B to reflect the change in P and go to Loop

---

The second refinement method in Prodigy, called protection, is similar to the confrontation step in Buridan and is not used in the example in this paper. I illustrate Weaver on the example used in the previous section, shown in figure 2.

Weaver initialises the plan and belief net, in which the goal g has zero probability of being true. It picks a high-probability initial state from the distribution, say the first outcome, and passes this to Prodigy with the goal g. Prodigy returns the one-step plan $[A]$, for which Weaver creates the belief net shown on the left in figure 5. This is evaluated to find that g is true with probability 0.525. The pair (p, initial) is identified to repair, and Weaver creates a branching plan based on p, passing the second possible initial state to Prodigy, and adding the new plan segment B, executed if p is false. The belief net for this plan, which is a linearisation of the one returned by Buridan, is shown in figure 5.

4.2 Search control and heuristics in Weaver

There are many ways Weaver could be improved. Its depth-first search technique can be costly, and its policy of calling Prodigy to plan to a goal state each time from different states leads to much duplication of planning effort. However it serves our purposes here to show

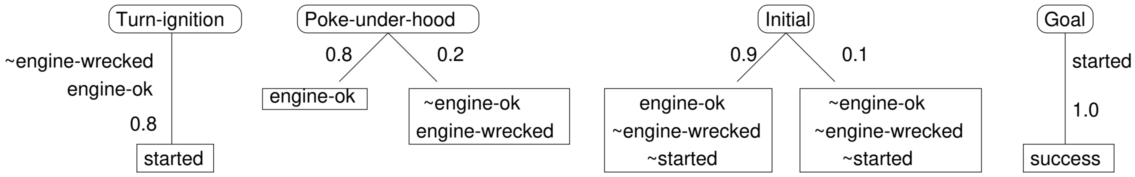


Figure 6: Operators in the car-starting domain

how powerful heuristics can be designed for a variety of probabilistic planners based on the footprint principle.

Weaver has several choice points in its search, including which of several potential failure pairs to address, whether to plan to avoid the failure or address it after its occurrence, which extra preconditions to add if avoiding the failure and which state to pick if planning after a failure's occurrence. By default, Weaver deals with failures that occur earlier before those that occur later. It prefers to plan to avoid a failure rather than to branch. If it branches, it creates a planning initial state for Prodigy by assuming the undesirable action outcome is the only one to occur. This set of defaults allows Weaver to solve the first example problem very quickly, and unaffected by the number n of identical operators. This is because its focus on early failures leads it to try different possible initial states. But it is merely fortunate that these heuristics lead quickly to the solution. If the initial state distribution gave outcome 1 a probability of 0.9 rather than 0.5, the shorter plan with probability above 0.95 would be to repeat action A, and Weaver would not immediately find this plan.

I illustrate a heuristic that can improve rates of convergence to the goal on a second domain, the "car-starting domain". The operators for this domain are shown in figure 6, and the goal is to achieve started with probability at least 0.95. In addition, the literal engine-ok is not observable, meaning that a branching plan based on this test is not allowed.

There is one plan of length 4 that achieves the goal with the required probability: turn the ignition twice, and if the car has not started, poke under the hood and then turn the ignition once more. If the ignition key is turned only once before the hood is opened, the problem cannot be solved because with too high a probability the car is wrecked[2], so Weaver's default heuristic will not work. If the agent never pokes under the hood the threshold probability cannot be reached.

I focus on the choice point of the planning initial state that Weaver passes to Prodigy when creating a branching plan, and derive a heuristic from the footprint principle . Recall that in this case, Weaver has chosen a literal l that has a low probability of taking a value required for the current plan, and an action that can lead to an undesired value for l. The default heuristic creates the state that would be produced if the operators in the plan gave the outcome nominated by Prodigy up to the suspect action, which then takes the most likely outcome that produces an undesired value for l.

I take a Bayesian view of the footprint principle described in section 2.2 and replace the default heuristic by one that attempts to create a likely state given that the original plan fails in a way predicted by l and a. The initial state distribution P_I is taken as a prior distribution, and the posterior, given that a produces an undesired value for l, is computed by propagating the undesired outcome as evidence in the belief net. Then one of the most likely states from the posterior distribution is chosen, and the plan simulated, each time picking one of the most likely outcomes given the new distribution.

When Weaver is run with the default heuristic, it begins solving the start-car problem with the plan turn-ignition, and then attempts to create a branching plan based on the test

[2]This is only a slight exaggeration of my skills as a mechanic.

started after the action is executed. With the default heuristic, Prodigy plans for a state with engine-ok true, producing the branching plan turn-ignition ; [started ? [] | turn-ignition]. Next Weaver creates a branching plan based on the test started after the second time the ignition is turned. The default heuristic produces exactly the same state, and Weaver appears caught in an endless loop[3], producing a plan whose probability of success asymptotically approaches 0.9.

With the footprint heuristic for choosing a state, Weaver starts in the same way, and tries to improve the plan [turn-ignition] when started is false. Now the probability of the initial state having engine-ok given that started is false drops from 0.9 to roughly 0.64. Since this is still the most likely state it is chosen and the new plan involves two turns of the ignition key as before. When Weaver creates a state for Prodigy given that the car fails to start twice, however, the probability of the initial state with engine-ok drops from 0.64 to 0.26, and it becomes more likely that the engine is broken. This state is chosen, the current plan is simulated on it, and Prodigy returns the plan poke-under-hood ; turn-ignition. The branching plan that results has probability just over 0.95 of success, and is returned.

In general, propagating evidence through the Bayes net to compute the posterior state distribution could take time exponential in the size of the net. In practice, updates are typically fast since the net can often be constructed to avoid large cycles and nodes with many parents. In other cases, a heuristic based on approximating the posterior distribution may still prove valuable.

5 Discussion

I have demonstrated how the footprint principle can be used to derive heuristics for two different probabilistics planners. Following this principle, one looks for heuristics that seek maximally disjoint refinements to the plan, defined in terms of the set of plan traces introduced in section 2.1. This approach can be usefully interpreted in terms of sets of traces, as shown for Buridan, or as using a Bayesian posterior probability distribution, as shown for Weaver.

Most of the previous work on probabilistic planning has not paid much attention to planning heuristics. Kushmerick, Hanks and Weld briefly discuss the possibility of using information from the plan evaluator to guide the search in [5], but do not mention specific heuristics. Goldman and Boddy discuss expanding the plan node with highest probability in their ϵ-safe planners [4]. There are interesting relations between this work and machine learning research on "boosting" learning algorithms [10]. Boosting essentially improves the performance of a learning algorithm by manipulating the probability distribution of learning examples to give a higher weight to those not covered by the algorithm's current output, a similar approach to the Bayesian view of the footprint principle.

The heuristics described in the previous two sections look quite different, but this appearance is mostly due to the difference in the way Buridan and Weaver organise their search. Buridan orders whole plans for subsequent refinement based on its ranking function, while Weaver requires choices to be made about different ways to refine a current plan. Both heuristics aim to maximise the amount by which a refinement applies to an area of the state space not already covered, although both are approximations. They in turn can be viewed as attempts to maximise the expected utility of the refinement to the plan, in the spirit of Russel and Wefald [9], under a neutral assumption of the relative difficulties of planning in the different parts of the state space.

[3] This is the behaviour with depth-first search, and it can be avoided using a scheme based on depth-first iterative deepening.

The heuristic used in Weaver is based on a global probability update, while the one used in Buridan makes a local estimation. If the initial state distribution in the first example problem were changed so that the only two-step plan was $[A; A]$, Weaver's heuristic would still lead it directly to the correct solution, while Buridan's would not. This increased sophistication has an accompanying cost in propagating the evidence through the Bayes net. It would be harder to produce an equivalent heuristic for Buridan because its ranking functions are applied to plans with unlinked triggers, while Weaver's are only applied to whole plans.

For each planner I have concentrated on only one aspect of the planner's search space, and the heuristics may of course make bad decisions in more complicated scenarios. Heuristics based on the footprint principle can also be found for at least some of the other choice points for each planner, and the combination of these should increase each planner's robustness.

The success of heuristics based on the footprint principle in two very different probabilistic planners makes it likely that it will be widely applicable in probabilistic planners. As well as existing planners, such as the ϵ-safe planners of Goldman and Boddy [4], I anticipate applying the footprint principle to planners that can represent initial state distributions and outcomes of actions that are composed of independent sources of uncertainty efficiently. Consider for example an action that can add three different literals l_i, each of whose probability of being added depends on a different literal l'_i, $1 \leq i \leq 3$. Although each of the l_i is added independently of the others, conditional on the l'_i, both Buridan and Weaver need to represent this action with eight different effect distributions, and would need to consider each separately in a plan to achieve l_i. An efficient planner for such a domain would need to reason about sets of states or alternatively about the probabilities of separate literals, using techniques like the Bayesian updating of probabilities across branching plans developed here for Weaver, and those based on the coherence principle to ensure the plans developed had a high-probability footprint. Thus principles such as these can be expected to become increasingly important for probabilistic planners in the future.

Acknowledgements

I'm grateful for conversations with Jaime Carbonell, Manuela Veloso and Steve Hanks, and for the comments of the anonymous reviewers.

References

[1] Jim Blythe. Planning with external events. In Ramon Lopez de Mantaras and David Poole, editors, *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, pages 94–101, Seattle, WA, July 1994. Morgan Kaufmann.

[2] Craig Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: structural assumptions and computational leverage. In *Proc. European Workshop on Planning*, Assissi, Italy, September 1995. IOS Press.

[3] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Mei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University, June 1992.

[4] Robert P. Goldman and Mark S. Boddy. Epsilon-safe planning. In Ramon Lopez de Mantaras and David Poole, editors, *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, pages 253–261, Seattle, WA, July 1994. Morgan Kaufmann.

[5] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. Technical Report 93-06-03, Department of Computer Science and Engineering, University of Washington, June 1993.

[6] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic least-commitment planning. In *Proc. Twelfth National Conference on Artificial Intelligence*, pages 1073–1078. AAAI Press, 1994.

[7] Todd Michael Mansell. A method for planning given uncertain and incomplete information. In David Heckerman and Abe Mamdani, editors, *Uncertainty in Artificial Intelligence*, volume 9, pages 350–358, Washington, D.C., July 1993. Morgan Kaufmann.

[8] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proc. Ninth National Conference on Artificial Intelligence*, pages 634–639. AAAI Press, 1991.

[9] Stuart Russell and Eric Wefald. *Do the Right Thing*. MIT Press, 1991.

[10] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.