Journal of Artificial Intelligence Research 57 (2016) 229–271 Submitted 03/16; published 10/16

Goal Probability Analysis in MDP Probabilistic Planning: Exploring and Enhancing the State of the Art

Marcel Steinmetz STEINMETZ@CS.UNI-SAARLAND.DE
Jörg Hoffmann HOFFMANN@CS.UNI-SAARLAND.DE
Saarland University,
Saarland Informatics Campus,
Saarbrücken, Germany
Olivier Buffet OLIVIER.BUFFET@LORIA.FR
INRIA / Université de Lorraine / CNRS,
Nancy, France

Abstract

Unavoidable dead-ends are common in many probabilistic planning problems, e.g. when actions may fail or when operating under resource constraints. An important objective in such settings is *MaxProb*, determining the maximal probability with which the goal can be reached, and a policy achieving that probability. Yet algorithms for MaxProb probabilistic planning are severely under-explored, to the extent that there is scant evidence of what the empirical state of the art actually *is*. We close this gap with a comprehensive empirical analysis. We design and explore a large space of heuristic search algorithms, systematizing known algorithms and contributing several new algorithm variants. We consider MaxProb, as well as weaker objectives that we baptize *AtLeastProb* (requiring to achieve a given goal probabilty threshold) and *ApproxProb* (requiring to compute the maximum goal probability up to a given accuracy). We explore both the general case where there may be 0-reward cycles, and the practically relevant special case of acyclic planning, such as planning with a limited action-cost budget. We design suitable termination criteria, search algorithm variants, dead-end pruning methods using classical planning heuristics, and node selection strategies. We design a benchmark suite comprising more than 1000 instances adapted from the IPPC, resource-constrained planning, and simulated penetration testing. Our evaluation clarifies the state of the art, characterizes the behavior of a wide range of heuristic search algorithms, and demonstrates significant benefits of our new algorithm variants.

1. Introduction

Many probabilistic planning problems contain *unavoidable dead-ends* (e.g. Kolobov, Mausam, Weld, & Geffner, 2011; Teichteil-Königsbuch, Vidal, & Infantes, 2011; Kolobov, Mausam, & Weld, 2012; Teichteil-Königsbuch, 2012), i.e., no policy guarantees to eventually, under all circumstances, attain the goal. Examples are planning under resource constraints or a limited budget, or situations where actions may fail and we will eventually run out of options. One important objective then is *MaxProb*, determining the maximal probability with which the goal can be reached (and identifying a policy achieving that probability). MaxProb also partly underlies the International Probabilistic Planning Competition (IPPC) (Younes, Littman, Weissman, & Asmuth, 2005; Bryce & Buffet, 2008; Coles, Coles, García Olaya, Jiménez, Linares López, Sanner, & Yoon, 2012), when planners are evaluated by how often they reach the goal in online policy execution. (The time limit in the IPPC setting mixes MaxProb with a bias towards policies reaching the goal quickly. This also relates to the proposals by Kolobov et al., 2012 and Teichteil-Königsbuch, 2012, asking for the cheapest policy among those maximizing goal probability, and to the proposal by Chatterjee, Chmelik, Gupta, & Kanodia, 2015, 2016, asking for the cheapest policy ensuring that a target state is reached almost surely in a partially observable setting.)

We consider MDP-based probabilistic planning, with factored models (probabilistic extensions of STRIPS) whose state spaces may be too large to build explicitly. We focus on the optimal offline setting, i.e., solving MaxProb exactly. While this setup and objective certainly is relevant, there has been little work towards developing solvers. The main effort was made by Kolobov et al. (2011), which we discuss in detail below. Hou, Yeoh, and Varakantham (2014) consider several variants of topological VI (Dai, Mausam, Weld, & Goldsmith, 2011), solving MaxProb but necessitating to build the entire reachable state space. Other works addressing goal probability maximization do not aim at guaranteeing optimality (e.g. Teichteil-Königsbuch, Kuter, & Infantes, 2010; Camacho, Muise, & McIlraith, 2016).

MDP heuristic search (Barto, Bradtke, & Singh, 1995; Hansen & Zilberstein, 2001; Bonet & Geffner, 2003b; McMahan, Likhachev, & Gordon, 2005; Smith & Simmons, 2006; Bonet & Geffner, 2006) has the potential to find optimal policies without building the entire state space, but Kolobov et al. (2011) are the only authors addressing optimal MaxProb through heuristic search. Part of the reason for this lack of research on heuristic search for MaxProb are the following two major obstacles. First, while MDP heuristic search has been successful in expected-cost minimization, it suffers from a lack of admissible (upper-bounding) heuristic estimators of goal probability. The best known possibility is to detect *dead-ends* and set their initial heuristic estimate to 0, using the trivial upper bound 1 elsewhere. Second, MaxProb does not fit the stochastic shortest path (SSP) framework (Bertsekas, 1995), due to 0-reward cycles. As pointed out by Kolobov et al. (2011), MaxProb is equivalent to a non-discounted reward maximization problem, where non-goal cycles receive 0 reward and thus improper policies do not accumulate reward $-\infty$.

To address the second problem, Kolobov et al. (2011) devised the *FRET (find, revise, eliminate traps)* framework, which admits heuristic search, yet requires several iterations of complete searches. In between heuristic search iterations, FRET eliminates 0-reward cycles (traps). FRET iterates until no more such cycles persist. Kolobov et al.'s contribution is mainly theoretical – considering not only MaxProb but a much larger class of *generalized SSPs* – and their empirical evaluation serves merely as a proof of concept. They experiment with a single domain (ExplodingBlocks), and run only one configuration of search (LRTDP, Bonet & Geffner, 2003b), with one possibility for dead-end detection and thus non-trivial initial heuristic estimates (SixthSense, Kolobov, Mausam, & Weld, 2010). This does outperform value iteration (VI), but the dead-end detection is not used in VI, and it remains unclear to what extent the improvement is due to the actual heuristic search, rather than the state pruning itself.

In summary, heuristic search for MaxProb is challenging, and has only been addressed by Kolobov et al. (2011), with very limited experiments. Given this:

(i) *What is actually the empirical state of the art in heuristic search for MaxProb? Are there other known algorithms, or variants thereof, that work better?*

We explore a large design space of such algorithms, and show that, indeed, some variants work much better.

(ii) *What about simpler yet still relevant special cases, and weaker objectives, that may be easier to solve?*

There are indeed practically relevant cases that do not necessitate FRET, and weaker objectives that enable what we will refer to as *early termination*.

To elaborate first on (ii): If the state space of the planning task at hand is acyclic, then clearly FRET is not needed: there are no cycles – so in particular no 0-reward cycles – and the state space is finite, so that any execution will end in a (goal or non-goal) absorbing state; this implies that we are within the realm of SSPs. This special case is, however, still practically relevant. As an illustration, acyclic state spaces occur even in a standard IPPC benchmarks, namely in the TriangleTireworld domain where moves can only be made in one direction. More importantly, planning with a limited action-cost budget, *limited-budget planning*, has acyclic state spaces when action costs are non-0, strictly decreasing the remaining budget. A similar class of scenarios is where every action consumes a non-0 amount of some non-replenishable resource. Another example are recently proposed models of simulated *penetration testing*, as per Hoffmann (2015). The MDP there models a network intrusion from the point of view of an attacker. The state space is acyclic because each exploit can be attempted at most once (trying the same exploit again on the same network configuration would yield the same outcome). States thus need to remember the remaining action set, and every action application strictly reduces that set.

Regarding weaker objectives: As alternatives to MaxProb, it is reasonable to ask whether the maximum goal probability exceeds a given threshold θ, or to require computing the maximum goal probability up to a given accuracy δ. We refer to these objectives as *AtLeastProb* and *ApproxProb* respectively.[1] For example, in penetration testing, AtLeastProb naturally assesses the level of network security: Can an attacker reach a target host with probability greater than a given security margin? E.g., can a customer data server be compromised with probability greater than 0.01?

AtLeastProb and ApproxProb allow early termination based on maintaining both, a lower (pessimistic) bound V^L and an upper (admissible/optimistic) bound V^U. This is especially promising in AtLeastProb, where we can terminate if the lower bound already is good enough ($V^L \geq \theta$), or if the upper bound already proves infeasibility ($V^U < \theta$). Good anytime behavior, on either or both bounds, translates into early termination.

Let us now elaborate on (i), exploring the state of the art and beyond. We design an algorithm space characterized by:

(a) *Search algorithm.* We design variants of AO* (Nilsson, 1971), LRTDP (Bonet & Geffner, 2003b), and depth-first oriented heuristic searches (Bonet & Geffner, 2003a, 2006), maintaining upper and lower bounds for early termination.

(b) *FRET.* We design a new variant of FRET better suited to problems with uninformative initial upper bounds.

(c) *Bisimulation reduction.* We design a new probabilistic-state-space reduction method, via bisimulation relative to the all-outcomes determinization (e.g. Bonet & Geffner, 2003b; Yoon, Fern, & Givan, 2007; Little & Thiebaux, 2007).

1. AtLeastProb relates to MDP model-checking, where one typically wants to validate that a given PCTL (Probabilistic Computation Tree Logic) formula is valid with some probability (Baier, Größer, Leucker, Bollig, & Ciesinski, 2004; Kwiatkowska, Parker, & Qu, 2011a; Kwiatkowska, Norman, & Parker, 2011b). It also relates to Constrained MDPs (Altman, 1999), as enforcing a minimum success probability could be expressed through a constraint on a particular quantity. Chance-Constrained POMDPs (Santana, Thibaux, & Williams, 2016) are different from AtLeastProb as their constraint is on the probability to remain in safe states, not to reach goal states.

(d) *Dead-end pruning method.* We employ classical-planning heuristic functions for dead-end detection in probabilistic planning, via the all-outcomes determinization, as previously done by Teichteil-Königsbuch et al. (2011). This is especially promising in limited-budget planning, where we can prune a state s if an admissible classical-planning estimate exceeds the remaining budget in s.

(e) *Node selection strategy.* We design a comprehensive arsenal of simple strategies, biasing tie breaking in action and state selection in manners targeted at fostering early termination.

We implemented all these techniques within Fast Downward (FD) (Helmert, 2006), thus contributing, as a side effect of our work, an ideal implementation basis for exploiting classical-planning heuristic search techniques in MDP heuristic search.[2]

The algorithm dimensions (a) – (e) are orthogonal (excepting some dependencies, in particular that bisimulation reduction subsumes dead-end pruning). We explore the behavior of the resulting design space on a large benchmark suite we design for that purpose. The suite includes domains from the IPPC, resource-constrained planning, and penetration testing, each with with a limited-budget version and an unlimited-budget version. The suite comprises 1089 benchmark instances in total.[3] Amongst other things, we observe:

- Heuristic search yields substantial benefits, even with the *trivial* admissible heuristic setting the initial estimate to 1 everywhere (+9% total coverage across all benchmarks), more so with admissible heuristics based on dead-end detection (+12%).
- Early termination yields substantial benefits (e.g. for AtleastProb +8% with $\theta = 0.2$ and +7% with $\theta = 0.9$).
- Our FRET variant yields dramatic benefits (+32% total coverage on the cyclic benchmarks).
- Bisimulation reduction yields an optimal MaxProb solver that excells in TriangleTireworld, even surpassing Prob-PRP (Muise, McIlraith, & Beck, 2012; Camacho et al., 2016) – and this not only in the standard version where the goal can be achieved with certainty and hence Prob-PRP is optimal, but also in the limited-budget version where that is not so.

On the side, we discover that landmarks compilation as per Domshlak and Mirkis (2015), employed for dead-end pruning in their oversubscription planning setting, is actually, on its own, equivalent to pruning against the remaining budget with a standard admissible landmark heuristic. This is relevant to our work because, otherwise, that compilation would be a canonical candidate also for dead-end pruning in our setting (indeed this is what we started out with in our investigation).

The paper is organized as follows. Section 2 describes our model syntax and semantics, for goal probability analysis with and without an action-cost budget limit. Section 3 specifies our search algorithm (a) and FRET variants (b). Section 4 describes our bisimulation reduction method (c). Section 5 describes the dead-end pruning methods (d), and Section 6 describes the node selection strategies (e). We present our experiments in Section 7, and we conclude in Section 8. There are two appendices giving additional technical details that we only sketch in the main text, Appendix B

2. The source code is available at `http://fai.cs.uni-saarland.de/downloads/fd-prob.tar.bz2`
3. The benchmark suite is available at `http://fai.cs.uni-saarland.de/downloads/ppddl-benchmarks-acyclic.tar.bz2` (acyclic cases) and `http://fai.cs.uni-saarland.de/downloads/ppddl-benchmarks-cyclic.tar.bz2` (cyclic cases).

regarding Domshlak and Mirkis' (2015) landmarks compilation, and Appendix A regarding depth-first oriented heuristic searches. [4]

2. MDP Models

We consider PPDDL-style models (Younes et al., 2005), more precisely probabilistic extensions of STRIPS. We employ two formalism variants, with and without a limited action-cost budget. We specify first the unlimited-budget version. Planning *tasks* are tuples $\Pi = (F, A, I, G)$ consisting of a finite set F of *facts*, a finite set A of *actions*, an *initial state* $I \subseteq F$, and a *goal* $G \subseteq F$. Each $a \in A$ is a pair $(pre(a), O(a))$ where $pre(a) \subseteq F$ is the *precondition*, and $O(a)$ is the finite set of *outcomes* o. Each $o \in O(a)$ is a tuple $(p(o), add(o), del(o))$ of *outcome probability* $p(o)$, *add list* $add(o) \subseteq F$, and *delete list* $del(o) \subseteq F$. We require that $\sum_{o \in O(a)} p(o) = 1$.

Given a task Π, its *state space* is a probabilistic transition system $(S, P, I, S_\top)$. Here, S is the set of *states*, each $s \in S$ associated with its set $F(s)$ of true facts. The initial state I is that of Π. The set of *goal states* $S_\top \subseteq S$ contains those s where $G \subseteq F(s)$. Transitions, and the *transition probability* function $P : S \times A \times S \mapsto [0, 1]$, are defined as follows. Action a is *applicable* to state s if $pre(a) \subseteq F(s)$ and $s \notin S_\top$ (goal states are absorbing, see also below). By $A[s]$ we denote the set of actions applicable in s. Given s, $a \in A[s]$, and an outcome $o \in O(a)$, by $s[\![o]\!]$ we denote the result of outcome o in s, i.e., $F(s[\![o]\!]) := (F(s) \cup add(o)) \setminus del(o)$. We define $P(s, a, t) := p(o)$ if a is applicable to s and $t = s[\![o]\!]$.[5] Otherwise, we define $P(s, a, t) := 0$ (there is no transition). *Absorbing states* are those with no outgoing transitions (no applicable actions). The set of non-goal absorbing states – *lost states* – is denoted $S_\perp$.

For limited-budget planning, we extend the above as follows. A *limited-budget task* is a tuple $\Pi = (F, A, I, G, b)$, as above but now including also a *budget* $b \in \mathbb{R}^+_0$, and associating each action outcome o with a *cost* $c(o) \in \mathbb{R}^+_0$. In addition to their true facts $F(s)$, states s are now also associated with their *remaining budget* $b(s) \in \mathbb{R}$. States with negative remaining budget $b(s) < 0$ are legal and may occur, but they are lost, $s \in S_\perp$, due to the following definitions of goal states, action applicability, and transitions. The goal states $s \in S_\top$ are those where $G \subseteq F(s)$ *and* $b(s) \geq 0$, i.e., we must reach the goal with ≥ 0 remaining budget. The actions a applicable to s are those where $pre(a) \subseteq F(s)$ *and* at least one outcome fits within the remaining budget, i.e., there exists $o \in O(a)$ so that $c(o) \leq b(s)$. In the outcome states $s[\![o]\!]$, the outcome's cost is deduced from the remaining budget, i.e., $b(s[\![o]\!]) := b(s) - c(o)$.

A few notes are in order regarding limited-budget planning. If $c(o) > 0$ for all o, then the state space – viewed as a directed graph with an arc (s, t) whenever there is an action mapping s into t with non-0 probability – is acyclic because every transition strictly reduces the remaining budget. The state space is infinite due to the continuous state variable $b(s)$, but its reachable part (which our algorithms consider) is finite. Note further that the remaining budget is local to each state. If some states in a policy violate the budget, other parts of the policy (even other outcomes of the same action) can still continue trying to reach the goal. This differs from constrained MDPs (Altman, 1999), where the budget bound is applied globally to the expected cost of the policy. Also note

4. This paper is an extension of a previous conference paper (Steinmetz, Hoffmann, & Buffet, 2016). We cover a larger space of algorithms (now including depth-first oriented heuristic searches), provide comprehensive explanations and discussions, and present our experiments in detail.
5. We assume here that each $o \in O(a)$ leads to a different outcome state. This is just to simplify notation (our implementation does not make this assumption).

that, while a single budget is considered here for the sake of simplicity, our framework and results straightforwardly extend to models with multiple budget variables.

Limited-budget planning has been explored in a deterministic oversubscription setting, the objective being to maximize the reward from achieved (soft) goals subject to the budget (Domshlak & Mirkis, 2015). A classical-planning variant would relate to resource-constrained planning (e.g. Haslum & Geffner, 2001; Nakhost, Hoffmann, & Müller, 2012; Coles, Coles, Fox, & Long, 2013) with a single consumed resource. Our probabilistic variant here has been previously considered only by Hou et al. (2014). Prior work on probabilistic planning with resources (e.g. Marecki & Tambe, 2008; Meuleau, Benazera, Brafman, Hansen, & Mausam, 2009; Coles, 2012) has often assumed limited budgets and non-0 consumption, but has dealt with uncertain-continuous resource consumption, in contrast to the discrete and fixed budget consumed by action costs.

Though relatively restricted, limited-budget probabilistic planning is quite natural. Decision making is often constrained by a finite budget. Furthermore, non-0 costs are often reasonable to assume. This applies to, for example, penetration testing. Problems asking to achieve a goal within a given number of steps, e.g. finite-horizon goal probability maximization, are a special case.

Let us now define solutions to our planning tasks, as well as the objectives we wish these to achieve. A *policy* is a partial function $\pi : S \setminus (S_\top \cup S_\bot) \mapsto A \cup \{*\}$, mapping each non-absorbing state s within its domain either to an action applicable in s, or to the *don't care* symbol $*$. That symbol will be used (only) by policies that already achieve sufficient goal probability elsewhere, so do not need to elaborate on how to act on s and its descendants. That is, we still require *closed* policies (see below), and we use $*$ to explicitly indicate special cases where actions may be chosen arbitrarily. Formally, $\pi(s) = *$ extends the domain of π by picking, for every $t \notin S_\top \cup S_\bot$ reachable from s and where $\pi(t)$ is undefined, an arbitrary action a applicable in t and setting $\pi(t) := a$.

A policy π is *closed for state* s if, for every state $t \notin S_\top \cup S_\bot$ reachable from s under π, $\pi(t)$ is defined. π is *closed* if it is closed for the initial state I. π is *proper* if, from every state s on which π is defined, π eventually reaches an absorbing state with probability 1.[6]

Following Kolobov et al. (2011), we formulate goal probability as maximal non-discounted expected reward where reaching the goal gives reward 1 and all other rewards are 0. The value $V^\pi(s)$ of a policy π closed for state s then is:

$$V^\pi(s) = \begin{cases} 1 & s \in S_\top \\ 0 & s \in S_\bot \\ \sum_t P(s, \pi(s), t) V^\pi(t) & \text{otherwise} \end{cases} \tag{1}$$

The *optimal* value of state s is

$$V^*(s) = \max_{\pi:\pi \text{ closed for } s} V^\pi(s) \tag{2}$$

Observe here that, in difference to Kolobov et al. who consider problems more general than MaxProb, we don't need to exclude improper π from this maximization. This is because there are no negative rewards, i.e., policies cannot gain anything from infinite cycles.

Given a value function V (any function mapping states to $\mathbb{R}$), the Bellman *update* operator is defined, as usual, through maximization over actions relative to the current values given by V:

6. Keep in mind here that the absorbing states in our setting are $S_\top \cup S_\bot$, i.e., goal states *and* lost states. While an SSP policy can only be considered as valid when all executions end up in a goal state – because finding a shortest path implies that a path exists – a MaxProb policy is valid when all executions end up in an absorbing (goal or non-goal) state – executions may fail, but need to always terminate.

$$V(s) := \begin{cases} 1 & s \in S_\top \\ 0 & s \in S_\perp \\ \max_{a \in A[s]} \sum_t P(s,a,t)V(t) & \text{otherwise} \end{cases} \tag{3}$$

The difference between $V(s)$ prior to the update, and its updated value according to the right-hand side, is called the Bellman *residual*.

The *greedy policy* π on a value function V selects in each non-absorbing state an action obtaining the maximum in the right-hand side of this equation (note that the greedy policy is unique only up to tie-breaking). We will refer to the state space subgraph induced by those states reachable from I using such a greedy policy π as the *π-greedy graph*. By the *V-greedy graph*, we will refer to the state space subgraph induced by those states reachable from I by *any* greedy policy on V, i.e., allowing in each state to choose any action greedy on V.

For acyclic state spaces, every run ends in an absorbing state in a finite number of steps, so we are facing an SSP problem (subject to our definition of absorbing states, cf. above) and the Bellman update operator has the unique fixed point V^*, which it converges to from any initial V. For cyclic state spaces, as pointed out by Kolobov et al. (2011), the Bellman update operator may have multiple sub-optimal fixed points, and updates from an optimistic (upper-bound) initialization V are not guaranteed to converge to the optimum V^*. One can either use a pessimistic (lower-bound) initialization V, from which the updates are guaranteed to converge to V^*; or one can use Kolobov et al.'s FRET method described earlier.

We consider three different *objectives* (algorithmic problems) for goal probability analysis:

MaxProb: Find an optimal policy, i.e., a closed π s.t. $V^\pi(I) = V^*(I)$.

AtLeastProb: Find a policy guaranteeing a user-defined goal probability threshold $\theta \in [0, 1]$, i.e., a closed π s.t. $V^\pi(I) \geq \theta$. (Or prove that such π does not exist.)

ApproxProb: Find a policy optimal up to a user-defined goal probability accuracy $\delta \in [0, 1]$, i.e., a closed π s.t. $V^*(I) - V^\pi(I) \leq \delta$.

We now define our algorithm family addressing these problems. We cover search algorithms, bisimulation reduction, dead-end pruning, and node selection strategies, in this order.

3. Search Algorithms

We use value iteration (VI) as a baseline. We design variants of AO* and LRTDP, as well as a family of depth-first oriented heuristic searches, systematizing algorithm parameters underlying improved LAO* (here: ILAO*) (Hansen & Zilberstein, 2001), heuristic dynamic programming (Bonet & Geffner, 2003a), and learning depth-first search (Bonet & Geffner, 2006). We furthermore design a variant of FRET better suited to problems with uninformative initial upper bounds.

3.1 VI

As a pre-process to VI, we make one forward pass building the reachable state space (actually its pruned subgraph, see Section 5). We initialize the value function pessimistically, simply as 0 everywhere. For acyclic cases, we then perform a single backward pass of Bellman updates, starting at absorbing states and updating children before parents, thus computing the optimal value function while updating every state exactly once.

procedure *GoalProb-AO**
initialize Θ to consist only of I; *Initialize*(I)
loop do
  if [MaxProb: $V^L(I) = 1$]
    [AtLeastProb: $V^L(I) \geq \theta$]
    [ApproxProb: $V^L(I) \geq 1 - \delta$ or $V^U(I) - V^L(I) \leq \delta$] **then**
    return π^L **endif** /* early termination (positive) */
  if [AtLeastProb: $V^U(I) < \theta$] **then**
    return "impossible" **endif** /* early termination (negative) */
  if ex. leaf state $s \notin S_\top \cup S_\perp$ in Θ reachable using π^U **then**
    select such a state s
  else return π^U **endif** /* regular termination */
  for all a and t where $P(s, a, t) > 0$ **do**
      if t not already contained in Θ **then**
        insert t as child of s into Θ; *Initialize*(t)
      else insert s as a new parent of t into Θ
      endif
  endfor
  BackwardsUpdate(s)
endloop
procedure *Initialize*(s):
$V^U(s) := \begin{cases} 0 & s \in S_\perp \\ 1 & \text{otherwise} \end{cases}$
$V^L(s) := \begin{cases} 1 & s \in S_\top \\ 0 & \text{otherwise} \end{cases}$
if $s \notin S_\top \cup S_\perp$ **then** $\pi^L(s) := *$ **endif**

Figure 1: AO* search for MaxProb, AtLeastProb, and ApproxProb (as indicated), on acyclic state spaces. π^U is the current greedy policy on V^U, π^L is the current greedy policy on V^L. The *BackwardsUpdate*(s) procedure updates all of V^U, π^U, V^L, π^L. As states may have several parents in Θ, we first make a backwards sweep to collect the sub-graph $\Theta|_s$ ending in s (to update V^U and π^U, the greedy sub-graph on V^U suffices). Then we update $\Theta|_s$ in reverse topological order.

For the general/cyclic case, we assume a *convergence parameter* ϵ (likewise in all other algorithms addressing this case), and compute an ϵ*-consistent* value function, where the Bellman residual on every state is at most ϵ. For efficient value iteration, we employ *topological VI* as per Dai et al. (2011): we find the strongly connected components (SCC) of the state space, and handle each SCC individually, children SCCs before parent SCCs. VI on an SCC stops when every state is ϵ-consistent.

Dai et al. (2011) also introduce *focused* topological VI, which eliminates sub-optimal actions in a pre-process to obtain smaller SCCs. While this can be much more runtime-effective, it still requires building the entire state space. In our experiments, runtime/memory exhaustion during this process, i.e., during building the state space, was the *only* reason for VI failures. So we do not consider focused topological VI here.

3.2 AO*

For AO*, we restrict ourselves to the acyclic case, where the overhead for repeated value iteration fixed points, inherent in LAO* (Hansen & Zilberstein, 2001), disappears. (The ILAO* variant, where that issue has been addressed through a depth-first orientation, is covered as part of our depth-first oriented heuristic search family introduced in Section 3.4 below.)

Figure 1 shows pseudo-code for our *GoalProb-AO** variant. The algorithm incrementally constructs a subgraph Θ of the state space. The handling of duplicates is simple, identifying search nodes with states, as the state space is acyclic. For the same reason, simple backward updating suffices to maintain the value function. Adopting ideas from prior work (e.g. McMahan et al., 2005; Little, Aberdeen, & Thiébaux, 2005; Smith & Simmons, 2006; Kuter & Hu, 2007), we maintain two value functions, namely both an upper bound V^U and a lower bound V^L on goal probability.

For lack of heuristic estimators of goal probability, both value functions are initialized trivially, by 1 for V^U and by 0 for V^L, except for absorbing states where the exact value is known. (Dead-end detection, as a simple but non-trivial V^U initialization, will be discussed in Section 5.) Nevertheless, both bounds can be useful for search, through early termination (V^L and V^U), and through detecting sub-optimal parts of the state space (V^U). To observe the latter, note that, to refute an action a, it may suffice to reduce V^U *for just one of a's outcomes*. Hence, even for trivial initialization, V^U may allow to disregard parts of the search space, in the usual way of admissible heuristic functions. As we shall see, this kind of behavior occurs frequently in practice (as reflected by our benchmarks).

Regarding early termination, the lower bound enables *positive* early termination when we can already guarantee sufficient goal probability, namely 1 (MaxProb), θ (AtLeastProb), or $1 - \delta$ (ApproxProb). The upper bound enables *negative* early termination in AtLeastProb, when $V^U(I) < \theta$. In ApproxProb, clearly we can terminate when $V^U(I) - V^L(I) \leq \delta$. A relevant observation here is that the $V^L(I) = 1$ (MaxProb) and $V^L(I) \geq 1 - \delta$ (ApproxProb) criteria are redundant when maintaining an upper bound, i.e., for heuristic search: If $V^L(I) \geq 1 - \delta$, then trivially also $V^U(I) - V^L(I) \leq \delta$. If $V^L(I) = 1$, then there is a search branch achieving the goal with certainty, so $V^U(I) = 1$ there as well and search terminates regularly. In configurations not maintaining V^U, however, these criteria can be very useful to reduce search.

The correctness of GoalProb-AO* is easy to establish. By the standard properties of Bellman updates, at any point in time during the execution of the algorithm, and for any state s in Θ, we have that $V^L(s) \leq V^*(s) \leq V^U(s)$, i.e., V^L and V^U are lower respectively upper bounds on goal probability. Indeed, these bounds are *monotone* (Bertsekas & Tsitsiklis, 1996), precisely, V^L and V^U are exact on absorbing states, and satisfy $V^L(s) \leq \max_{a \in A[s]} \sum_t P(s,a,t) V^L(t)$ respectively $V^U(s) \geq \max_{a \in A[s]} \sum_t P(s,a,t) V^U(t)$ on non-absorbing ones. This is because V^L and V^U are initialized with functions trivially satisfying these properties, and these properties are invariant over Bellman updates on non-absorbing states (given monotonicity, V^L can only grow, while V^U can only decrease). Thanks to monotonicity, with the same arguments as given for LAO* (Hansen & Zilberstein, 2001), we get that V^U converges to V^* in finite time on the π^U-greedy graph.

Finally, we need to prove that, in case of early termination returning π^L, the greedy policy π^L on V^L actually achieves what we want, i.e., (1) π^L is closed and (2) π^L provides sufficient goal probability, i.e., $V^{\pi^L}(I) \geq V^L(I)$. For (1), π^L is always a closed policy, because it applies the *don't care* symbol $*$ at the non-absorbing leaf states in Θ. (Note also that $*$ is applied *only* by π^L and *only* on those states.) For (2), we show that, for all states s, we have $V^{\pi^L}(s) \geq V^L(s)$. This claim is trivial for states s where $\pi^L(s) = *$, as these have never been updated so $V^L(s) = 0$. For other states s, the claim follows by a simple inductive reasoning over the maximal distance to an absorbing state in the π^L-greedy graph. For absorbing states s, we have $V^{\pi^L}(s) = V^L(s) = V^*(s)$, so the claim is trivially satisfied. In the induction step, we have $V^{\pi^L}(s) = \sum_t P(s, \pi^L(s), t) V^{\pi^L}(t)$ by definition of V^{π^L}, while, by the induction hypothesis, $V^{\pi^L}(t) \geq V^L(t)$ for all those states t

```
procedure GoalProb-LRTDP
Θ := {I}; Initialize(I)
loop do
  [early termination criteria exactly as in GoalProb-AO*]
  if I is not labeled as solved then
    LRTDP-Trial(I)
  else return π^U endif /* regular termination */
endloop
procedure LRTDP-Trial(s):
  P := empty stack
  while s is not labeled as solved do
    push s onto P
    if s ∈ S_⊤ ∪ S_⊥ then break endif
    [cyclic: if s is ε-consistent then break endif]
    for all a and t where P(s, a, t) > 0 do
       if t ∉ Θ then Initialize(t) endif
    endfor
    update V^U(s), π^U(s), V^L(s), π^L(s)
    s := sample t according to P(s, π^U(s), t)
  endwhile
  while P not empty do
    pop s from P
    [acyclic: if ¬ CheckSolved(s, 0) then break endif]
    [cyclic: if ¬ CheckSolved(s, ε) then break endif]
  endwhile
```

Figure 2: LRTDP for MaxProb, AtLeastProb, and ApproxProb, on acyclic or general (cyclic) state spaces. π^U is the current greedy policy on V^U, π^L is the current greedy policy on V^L. The *CheckSolved*(s, ϵ) procedure is exactly as specified by Bonet and Geffner (2003b). It visits states t reachable from s using π^U, initializing t if not previously visited, stopping at t if not ϵ-consistent. It then performs updates bottom-up, labeling t as solved iff all its descendants are ϵ-consistent. Our only change is to update V^L and π^L along with V^U and π^U.

where $P(s, \pi^L(s), t) > 0$, so in other words $V^{\pi^L}(s) \geq \sum_t P(s, \pi^L(s), t)V^L(t)$. By plugging in the definition of $\pi^L(s)$, and by using the monotonicity property, it is now easy to conclude that $V^{\pi^L}(s) \geq V^L(s)$, as desired.

3.3 LRTDP

Figure 2 shows pseudo-code for our *GoalProb-LRTDP* variant, applicable to the general case (cyclic as well as acyclic problems). We assume that, in cyclic cases, the algorithm is run within the FRET framework. The main change to the original version of LRTDP consists in maintaining a lower bound in addition to the upper (optimistic) bound, and adding the same early termination criteria as in GoalProb-AO*. Correctness of early termination follows with the same arguments as before, i.e., $V^L(s)$ and $V^U(s)$ are monotone lower respectively upper bounds, and π^L is always a closed policy. Note that this is true even in the general/cyclic case, i.e., if early termination applies, then we can terminate the overall FRET process.

The only other change we make is an additional stopping criterion for trials in the cyclic case, namely if the current state s is ϵ-consistent. Kolobov et al. (2011) use this criterion to keep trials

procedure *GoalProb-DFHS*
$\Theta := \{I\}$
loop do
  [early termination criteria exactly as in GoalProb-AO*]
  if (`Label` **and** I is not labeled as solved)
    or (`VI` **and** π^U changed after running VI on the π^U-greedy graph) **then**
    DFHS-Exploration(I)
    clean visited-markers
  else return π^U **endif** /* regular termination */
endloop
procedure *DFHS-Exploration*(s):
  if $s \notin \Theta$ **then** *Initialize(s)* **endif**
  if $s \in S_\top \cup S_\perp$ **or** s is labeled solved **then**
    label s solved
    return $\perp$
  endif
  $flag := \perp$
  if `FW` **then**
    if $V^U(s)$ is not ϵ-consistent **then** $flag := \top$ **endif**
    update $V^U(s)$, $\pi^U(s)$, $V^L(s)$, and $\pi^L(s)$
    if `Consist` **and** $flag$ **then return** $\top$ **endif**
  endif
  mark s as visited
  foreach t with $P(s, \pi^U(s), t) > 0$ **do**
    if t has not been visited **then** $flag :=$ DFHS-Exploration$(t) \vee flag$ **endif**
  done
  if $flag$ **or** $\neg$`FW` **then**
    if $V^U(s)$ is not ϵ-consistent **then** $flag := \top$ **endif**
    update $V^U(s)$, $\pi^U(s)$, $V^L(s)$, and $\pi^L(s)$
  endif
  if `Label` **and** $\neg flag$ **then** label s solved **endif**
  return $flag$

Figure 3: Depth-First Heuristic Search (DFHS) for acyclic MaxProb, AtLeastProb, and ApproxProb. The cyclic version is shown in Appendix A and uses Tarjan's SCC procedure instead of depth-first search. `VI`, `Label`, `FW`, and `Consist` are Boolean algorithm parameters (see text). Recall that the π^U-greedy graph is the set of states reachable from I using the current greedy policy π^U. The $flag$ returned by *DFHS-Exploration* is used inside the recursion only (it is ignored in the top-level calls), to decide whether to backward-update a state if forward-updates are in use.

from getting trapped in 0-reward (non-goal) cycles. The criterion preserves the property that, upon regular termination, all states reachable using π^U are ϵ-consistent.[7]

In the cyclic case, the V^U fixed point found by LRTDP may be sub-optimal, so we have to use FRET. In the acyclic case, we use $\epsilon = 0$, and a single call to LRTDP suffices.

3.4 Depth-First Heuristic Search

We finally consider systematic heuristic searches (not based on trials like LRTDP) with a strong depth-first orientation. Intuitively, such an orientation is especially beneficial in our context as it is likely to lead to absorbing states, and thus to states with a non-trivial heuristic function initialization, quickly. We refer to such algorithms as *Depth-First Heuristic Search (DFHS)*. Known instances are ILAO* (Hansen & Zilberstein, 2001),[8] heuristic dynamic programming (HDP) (Bonet & Geffner, 2003a), and learning depth-first search (LDFS) (Bonet & Geffner, 2006). Their commonality lies in conducting depth-first searches (DFS) on the state-space subgraph defined by actions greedy on a current upper bound V^U, which is being updated backwards in DFS, until a termination criterion applies. The algorithms differ in how depth-first branches are terminated, how the overall algorithm is terminated, and in whether or not updates are also performed in the forward direction. Here, we systematize these parameters, obtaining a DFHS algorithm family containing the previous algorithms as family members.

Figure 3 gives a pseudo-code description of our DFHS algorithm family. For simplicity, the figure considers acyclic problems only. For cyclic problems, instead of DFS the algorithms use Tarjan's depth-first SCC algorithm (Tarjan, 1972), in order to detect the SCCs at the same time as doing the exploration and updates, as suggested by Bonet and Geffner (2003a). (Knowing the SCCs is required for correct solved-labeling in the general case.) The pseudo-code description of the DFHS algorithm family for the general (cyclic) case is given in Appendix A.

The algorithms search in the π^U-greedy graph. A variant would be to instead search the V^U-greedy graph. That variant, employed by LDFS, is not effective for goal probability analysis because V^U is 1 everywhere initially, and the V^U-greedy graph is the entire (dead-end pruned) reachable state space. We hence omit this option, and therewith LDFS, from our DFHS family (matters may change if better admissible heuristic functions are identified in future work, cf. Section 8).

All algorithms update values in the backward direction, when leaving a state. If the `FW` algorithm parameter is true, then value updates are done also in the forward direction, when entering a state. As that consistently yields (small) advantages empirically, we switch `FW` to true in all our algorithm configurations, except in the one corresponding to the known algorithm ILAO* which does not use this technique. Detecting whether the optimal solution has been found can be done in two ways: (1) `Label`, maintaining solved-labels while doing the DFS; or (2) `VI`, running value iteration on the π^U-greedy graph after DFS has terminated. In (1), π^U is optimal if the initial state is labeled solved. In (2), one can terminate if the greedy policy did not change during VI. If we do use forward updates, then (as we already check the Bellman residual anyway) we have the additional option `Consist` to stop the search at ϵ-inconsistent states, as opposed to stopping only at absorbing states. Overall, we run 5 different parameter settings for DFHS, overviewed in Table 1.

Correctness of early termination follows again with the same arguments as before. For the correctness of regular termination, we need to show that a fixed point policy is obtained, i.e., upon regular termination, *(*) the π^U-greedy graph contains no ϵ-inconsistent states*. This holds because all our algorithm variants fit Bonet and Geffner's (2003a) Find-and-Revise schema on a finite state space with a monotone optimistic bound, where (1) in each search iteration we find and update at

7. The updates during trials are, in difference to the original LRTDP formulation, not related to a trial-stopping guarantee in goal probability maximization. They just turn out to consistently yield (small) advantages empirically, so we keep them in here.
8. The brief description of ILAO* by Hansen and Zilberstein (2001) – and thus its depth-first orientation – can be subject to interpretation. Our design here follows that of Bonet and Geffner (2005) in their mGPT tool.

Acronym	Termination	FW?	Cons?	Known?
DFHS$_{VI}$	VI	no	no	yes: ILAO* (Hansen & Zilberstein, 2001)
DFHS$^{Fwd}_{VI}$	VI	yes	no	no: new variant
DFHS$^{FwdCons}_{VI}$	VI	yes	yes	no: new variant
DFHS$^{Fwd}_{Lab}$	Label	yes	no	no: new variant
DFHS$^{FwdCons}_{Lab}$	Label	yes	yes	yes: HDP (Bonet & Geffner, 2003a)

Table 1: Depth-First Heuristic Search (DFHS) family overview. We do not include LDFS (Bonet & Geffner, 2006) as, due to considering the V^U-greedy graph rather than the π^U-greedy graph, LDFS does not work well on MaxProb (see text).

least one ϵ-inconsistent state, until (2) condition (*) is met. Given the depth-first search (respectively Tarjan's algorithm, in the general case) it is clear that (1) holds true. Regarding (2), this is obvious when the `VI` termination option is used;[9] it holds for the `Label` termination option because a state is labeled solved only when all its descendant states in the π^U-greedy graph are ϵ-consistent.

As done in Table 1, we will usually omit the "GoalProb-" in algorithm names. Keep in mind though that our algorithms differ from the original ones, in particular in terms of early termination which depends on the objective MaxProb, AtLeastProb, or ApproxProb. To study the termination benefits of the lower vs. upper bound, we will switch each bound on and off individually. Where X denotes one of our search algorithms, we denote by $X|_U$ and $X|_L$ the variants of X maintaining only V^U respectively only V^L. We will sometimes write $X|_{LU}$ to make explicit that both bounds are used. Early termination criteria involving the non-maintained bound are disabled. For $X|_U$, this leaves just the negative criterion $V^U(I) < \theta$ in AtLeastProb; $X|_L$ still has positive criteria.

We test a version $X|_L$ only for X=AO*, as a canonical representative of (non-VI) blind search. In $AO^*|_L$, all non-absorbing leaf states in Θ are open (rather than only those reachable using π^U), and in case of regular termination we return π^L.

3.5 FRET

As previously hinted, Kolobov et al.'s (2011) FRET performs an iteration of complete searches. It starts with some upper-bound approximation V^U of V^*, which is continuously updated throughout the FRET process. Within each FRET iteration, a heuristic search algorithm runs until termination, i.e., until finding a fixed point policy. In between these iterations, FRET runs a *trap elimination* step, which finds all *traps* in the V^U-greedy graph. FRET forces the next search iteration to not include these traps. FRET terminates if the V^U-greedy graph does not contain a trap.

The trap elimination step works as follows. A trap is a subset T of non-absorbing states in which any greedy policy will remain indefinitely, i.e., all outgoing transitions in the V^U-greedy graph of any $s \in T$ lead to another trap state $t \in T$. A trap T is removed by collapsing T's states into a single state s_T. The incoming transitions of s_T are those incoming to any state of T, and its outgoing transitions are those transitions of T-states exiting T (note that these transitions are, by construction, not contained in the V^U-greedy graph).

This transformation obviously prevents T from occuring again in later iterations. It preserves V^* as the trap states have identical V^* values: as all trap states are non-absorbing and can reach each other, these states can reach each other with 0-reward transitions (note that this holds regardless of

9. Note that, in the acyclic case, full VI is not actually needed so the algorithm could be simplified. We leave it this way here, as used by ILAO* in the general case, for simplicity of presentation.

V^U, i.e., it holds also on parts of the state space where V^U has not yet converged). Because there is only a finite number of possible traps in the state space, FRET eventually finds a V^U whose V^U-greedy graph does not contain a trap. From that graph, a V^U-greedy policy π is extracted, which does not contain traps, hence is proper on the trap-collapsed state space, hence is optimal for that state space. An optimal policy for the original task can be constructed from π by acting, within collapsed traps, in a way so that the exit taken by π is eventually reached with certainty. (This is the correctness argument given by Kolobov, 2013.)

Our new variant of FRET differs from the original version only in terms of the state space subgraph considered: instead of the V^U-greedy graph, we use the π^U-greedy graph, i.e., we consider only the actions selected into the current greedy policy (cf. our discussion of DFHS above). We will refer to this design as ***FRET*-π^U**, and we will refer to Kolobov et al.'s (2011) design as ***FRET*-V^U**. It is easy to see that FRET-π^U is still correct. The arguments above remain intact as stated.

FRET-V^U potentially eliminates more traps in each iteration, and may hence require fewer iterations. Yet not all these traps may actually need to be eliminated (we might eventually find an optimal policy not entering them), and each trap elimination step may be much more costly. In particular, in goal probability analysis, FRET-V^U is typically ineffective because, similarly as discussed above for DFHS, in the first FRET step V^U often is 1 almost everywhere, and the V^U-greedy graph is almost the entire reachable state space. As we shall see, FRET-π^U clearly outperforms FRET-V^U.

4. State-Space Reduction via Determinized Bisimulation

Bisimulation is a known method to reduce state space size in MDPs/probabilistic planning (e.g. Dean & Givan, 1997). The idea essentially is to group equivalent sets of states together as block states, and then solve the smaller MDP over these block states. Here, we observe that this approach can be fruitfully combined with state-of-the-art classical planning techniques, namely *merge-and-shrink* heuristics (Dräger, Finkbeiner, & Podelski, 2009; Helmert, Haslum, Hoffmann, & Nissim, 2014), which allow to effectively compute a bisimulation over the *determinized* state space. Determinized-bisimilar states are bisimilar in the probabilistic state space as well, so this identifies a practical special case of probabilistic bisimulation given a factored (STRIPS-like) problem specification.

Let us spell this out in a little more detail. Given a task Π (with or without budget limit), a *probabilistic bisimulation* for Π is a partitioning $P = \{B_1, \dots, B_n\}$ of Π's state set S so that, for every B_i and B_j, every action a, and every $s, t \in B_i$, the following two properties are satisfied (Dean & Givan, 1997):

(i) a is applicable in s iff a is applicable in t; and

(ii) if a is applicable in s and t, then $\sum_{o \in O(a), s[\![o]\!] \in B_j} p(o) = \sum_{o \in O(a), t[\![o]\!] \in B_j} p(o)$.

Dean and Givan show that an optimal solution to the bisimulation of an MDP induces an optimal solution to the MDP itself. In other words, it suffices to work on the block states B_i.

Now, denote by Π^{det} the *all-outcomes determinization* of Π (e.g. Yoon et al., 2007; Little & Thiebaux, 2007), with a separate action a_o^{det} for every a and $o \in O(a)$, inheriting a's precondition and o's adds, deletes, and cost. A *determinized bisimulation* for Π is a partitioning $P = \{B_1, \dots, B_n\}$ of Π's states so that, for every B_i and B_j, every determinized action a_o^{det}, and every $s, t \in B_i$, the following two properties are satisfied (Milner, 1990; Helmert et al., 2014):

(a) a_o^{det} is applicable in s iff a_o^{det} is applicable in t; and

(b) if a_o^{det} is applicable in s and t, then $s[\![a_o^{det}]\!] \in B_j$ iff $t[\![a_o^{det}]\!] \in B_j$.

It is easy to see that such $\{B_1, \dots, B_n\}$ also is a probabilistic bisimulation for Π. Since an action a_o^{det} is applicable in a state s iff the corresponding action a of the original MDP is applicable in s, (a) directly implies (i). From (b), we know that for every action a applicable to s, t, and for each outcome $o \in O(a)$, we have $s[\![a_o^{det}]\!] \in B_j$ iff $t[\![a_o^{det}]\!] \in B_j$. This obviously implies (ii); it is more restrictive than needed as it insists on the *subset of outcomes* being the same on both sides, rather than only their *summed-up probability* being the same.

But how to compute a determinized bisimulation for Π? The naïve solution is to build the state space up front and then computing a determinized bisimulation on it. One can potentially do much better though, by using merge-and-shrink with the widely employed shrinking strategies based on bisimulation (Nissim, Hoffmann, & Helmert, 2011; Katz, Hoffmann, & Helmert, 2012; Helmert et al., 2014). In a nutshell, this algorithm framework constructs an abstraction by starting with a collection of abstractions each considering a single state variable only, then iteratively "merging" two abstractions (replacing them with their synchronized product) until only a single abstraction is left, and "shrinking" abstractions to a bisimulation thereof in between every merging step. As we shall see in the experiments, this often still incurs a prohibitive overhead, but it can be feasible, and lead to substantial state space size reductions. In some cases, it results in tremendous performance improvements.

5. Dead-End Pruning

We refer to states s where $V^*(s) = 0$, i.e., the goal cannot be reached at all from s, as *dead-ends*. If one detects such s via some *dead-end detection* technique, then one can treat s exactly like a lost state $S_\perp$ (except for setting $\pi^L(s) := *$ as we need to act on non-absorbing states). This constitutes a pruning method in itself, useful for any search algorithm, as the state space below s needs no longer be explored. Apart from this pruning itself, for the heuristic search algorithms, dead-end detection provides a non-trivial initialization of V^U, as we will initialize $V^U(s) = 0$ instead of $V^U(s) = 1$ if we detected s to be a dead-end. This more informed initial upper bound typically leads to additional search reductions.

But how to detect dead-ends? Kolobov et al. (2011) employ SixthSense (Kolobov et al., 2010), which learns dead-end detection rules by generalizing from information obtained using a classical planner. Here we instead exploit the power of classical-planning heuristic functions – readily available in our FD implementation framework – run on the all-outcomes determinization. This is especially promising in limited-budget planning, where we can use lower bounds on determinized remaining cost to detect states with insufficient remaining budget. Observe that this is natural and effective using admissible remaining-cost estimators, yet would be impractical using an actual classical planner (which would need to be optimal and thus prohibitively slow). In the unlimited-budget case, we can use any heuristic function able to detect dead-ends (returning ∞), which applies to most known heuristics. Indeed, merge-and-shrink heuristics have recently been shown to be extremely competitive dead-end detectors (Hoffmann, Kissmann, & Torralba, 2014).

To make this concrete, consider a state s in a task Π, and denote as before by Π^{det} the all-outcomes determinization of Π. Let h be a classical-planning heuristic function. If h guarantees to return ∞ only on dead-ends, and $h(s) = \infty$ on Π^{det}, then there exists no sequence of action outcomes achieving the goal from s, so $V^*(s) = 0$. If Π is a limited-budget task, h is admissible, and $h(s) > b(s)$, then we cannot achieve the goal from s within the budget, and thus also $V^*(s) = 0$.

We experiment with state-of-the-art heuristic functions, namely (a) an *admissible landmark heuristic* as per Karpas and Domshlak (2009), (b) *LM-cut* (Helmert & Domshlak, 2009), (c) several variants of merge-and-shrink heuristics, and (d) $h^{\max}$ (Bonet & Geffner, 2001) as a simple and canonical option. (a) turned out to perform consistently worse than (b), so we will report only on (b) – (d).

For limited-budget planning, we also considered adopting the problem reformulation by Domshlak and Mirkis (2015) for oversubscription planning, which reduces the budget b using landmarks and in exchange allows traversing yet unused landmarks at a reduced cost during search. It turns out, however, that pruning states whose reformulated budget is < 0 is equivalent to the much simpler method pruning states whose heuristic (a) exceeds the (original/not reformulated) remaining budget. The added value of Domshlak and Mirkis' reformulation thus lies, not in its pruning per se, but in its compilation into a planning language and the resulting combinability with other heuristics.

We give full details in Appendix B. To get an intuition why Domshlak and Mirkis' reformulation is, per se, equivalent to (a), assume for simplicity that L is a set of disjoint disjunctive action landmarks for the initial state, and assume that actions have unit costs. Say we prune s if its reduced budget, $b'(s)$, is < 0. The reduced initial budget is $b' := b - |L|$. The reduced costs allow applying member actions of yet non-used landmarks at 0 cost, where the non-used landmarks for a given search path are those $l \in L$ not touched by the path. Consider now some state s reached on path $\vec{a}$. Denote the non-used landmarks by L'. The cost saved on $\vec{a}$ thanks to the reformulation is exactly that of the used landmarks, $|L \setminus L'|$. Hence $b'(s) = b' - (|\vec{a}| - |L \setminus L'|) = (b - |L|) - |\vec{a}| + |L \setminus L'| = b - |\vec{a}| - |L'|$. So s is pruned in the reformulation, $b'(s) < 0$, iff $b - |\vec{a}| - |L'| < 0$ iff $b - |\vec{a}| < |L'|$. The latter condition, however, is exactly the pruning condition using the simple method (a) instead.

6. Node Selection Strategies

In all our algorithms, good anytime behavior on V^L and/or V^U may translate into early termination. We explore the potential of fostering this via (1) biasing the tie-breaking in the selection of "best" actions π^U greedy with respect to V^U, and (2) biasing, respectively, the outcome-state sampling during trials (LRTDP) and the choice of expanded leaf states (AO*). To be precise regarding the latter: as usual, we maintain "state open" flags in AO*, true if a state has open descendants within the π^U-greedy graph. We select the leaf state to expand by going forward from I using π^U, and if an action has more than one open outcome state t, we select a t best according to the bias (2). Note that (2) is not as relevant in DFHS, which in every iteration of DFS explores all outcomes anyhow. Hence, in DFHS, we use only the π^U tie-breaking criteria (1) explained in what follows.

We experimented with a variety of strategies. In what follows, where a strategy specifies one of (1) or (2) only, the other setting is as in the *default* strategy. That strategy corresponds to the commonly used settings. It uses arbitrary tie-breaking for (1), but in a fixed manner, changing $\pi^U(s)$ only if some other action becomes strictly better in s, as suggested by Bonet and Geffner (2003b) for LRTDP. There is no bias (2) on outcome states in AO* (an open outcome state is selected arbitrarily). Bias (2) in LRTDP is by outcome probability. We also tried this *most-prob-outcome bias* strategy in AO*, where the most likely open outcome state is selected.

The *h-bias* strategy prefers states with smaller h value, where the heuristic h is the same one used for dead-end pruning.[10] Specifically, for action selection tie-breaking (1), from those actions

10. We also experimented with a strategy using merge-and-shrink with determinized action costs set to the negated logarithm of outcome probability (compare e.g. Jimenez, Coles, & Smith, 2006). This is compelling in theory

a maximizing the optimistic expected goal probability $\sum_t P(s,a,t)V^U(t)$, we select an a minimizing the expected heuristic value $\sum_t P(s,a,t)h(t)$. The outcome-state bias (2) is obtained by renormalizing the weighed probabilities $\frac{1}{h(t)} * P(s,a,t)$, so we prefer high probability outcomes with small h value.

Inspired by BRTDP (McMahan et al., 2005), we experiment with a *gap-bias* strategy, biasing search towards states with large $V^U - V^L$ gaps. Precisely, for (1) we break ties in favor of actions a maximizing the expected gap $\sum_t P(s,a,t)[V^U(t)-V^L(t)]$, and for (2) we renormalize the weighed probabilities $[V^U(t) - V^L(t)] * P(s,a,t)$.

Inspired by common methods in classical planning (e.g. Hoffmann & Nebel, 2001; Helmert, 2006; Richter & Helmert, 2009), we experiment with a *preferred actions* strategy, which in (1) prefers to set $\pi^U(s)$ to an action a participating in a delete-relaxed determinized plan for s, if such a maximizing $\sum_t P(s,a,t)V^U(t)$ exists.

AO$^*|_{\mathrm{L}}$ is a special case, as we do not maintain an upper bound and thus there is no selection (1) of actions π^U greedy with respect to V^U. We apply node selection strategies for (2) directly to the set of (all) leaf states in the current search graph Θ. The default strategy is depth-first, the rationale being to try to reach absorbing states quickly. The h-bias strategy selects a deepest leaf with minimal h value, the preferred actions strategy selects a deepest open leaf reachable using only preferred actions. We furthermore experiment with a breadth-first strategy, just for comparison.

7. Experiments

We implemented all the algorithms in Fast Downward (FD) (Helmert, 2006), and ran experiments on an extensive suite of benchmarks.[11] In our evaluation we first summarize the results for acyclic benchmarks (where FRET is not needed), and then the ones for cyclic benchmarks (where FRET is needed).

7.1 Experiments Setup

We start with giving details of our implementation and describing the benchmark suite used for the experiments.

7.1.1 IMPLEMENTATION

As our model pertains to goal-directed MDPs with a limited number of (explicitly listed) outcomes per action, naturally we use PPDDL (Younes et al., 2005), rather than RDDL (Sanner, 2010; Coles et al., 2012), as the surface-level language. FD's pre-processes were extended to handle PPDDL, and we added support for specifying a (numeric) budget limit.

Given the FD implementation framework in contrast to previous works on optimal probabilistic planning, we implemented all algorithms from scratch. For FRET, we closely followed the original implementation, up to details not specified by Kolobov et al. (2011), based on personal communication with Andrey Kolobov. (Kolobov's original source code is not available anymore, which also plays a role in our state-of-the-art comparison, see next.)

because, then, a bisimulation-based heuristic corresponds to the exact goal probability of the best outcome sequence from a state. Yet, as already pointed out, computing such a heuristic is often infeasible.

11. The source code is available in an online appendix, and can be downloaded at http://fai.cs.uni-saarland.de/downloads/fd-prob.tar.bz2

Given the scant prior work on optimal goal probability analysis (cf. Section 1), the state of the art is represented by topological VI, by LRTDP$|_{\mathrm{U}}$ with dead-end pruning on acyclic problems, and by FRET-V^U using LRTDP$|_{\mathrm{U}}$ with dead-end pruning on cyclic problems. All these configurations are particular points in the space of configurations we explore, so the comparison to the state of the art is part of our comparison across configurations. The only thing missing here is the particular form of dead-end detection, which was SixthSense in the only prior work, by Kolobov et al. (2011). As SixthSense is a complex method and advanced dead-end pruning via heuristic functions is readily available in our framework, we did not re-implement SixthSense. Our discussion of cyclic problems in Section 7.3 below includes a detailed comparison of our results with those by Kolobov et al., on IPPC ExplodingBlocks which is the only domain Kolobov et al. considered.

Note that providing quality guarantees is an important property in this study. For this reason, and for the sake of clarity, we do not compare against unbounded suboptimal approaches, such as using an algorithm with a discounted criterion or assigning large finite penalties to dead-ends (Teichteil-Königsbuch et al., 2011; Kolobov et al., 2012).

Furthermore, as AtLeastProb is a special case of MDP model checking, one may wonder how probabilistic model checking tools, e.g. PRISM (Kwiatkowska et al., 2011b), would fare on that problem in planning benchmarks. We do not investigate that question here, which would entail a translation from PPDDL into a model checking language, which is non-trivial and makes a direct comparison – of algorithms taking different inputs – problematic. One may speculate that, given their focus on blind searches, model checking tools are inferior to our heuristic search approaches where those fare well; but that remains a question for future work.

7.1.2 BENCHMARK SUITE

Our aim being to comprehensively explore the relevant problem space, we designed a broad suite of benchmarks, 1089 instances in total, based on domains from the IPPC, resource-constrained planning, and penetration testing (*pentesting*).

From the IPPC, we selected those PDDL domains in STRIPS format, or with moderate non-STRIPS constructs easily compilable into STRIPS. This resulted in 10 domains from IPPC'04 – IPPC'08; we selected the most recent benchmark suite for each of these.

For resource-constrained planning, we adopted the NoMystery, Rovers, and TPP benchmarks by Nakhost et al. (2012), more precisely those suites with a single consumed resource (fuel, energy, money), which correspond to limited-budget planning.[12] We created probabilistic versions by adding uncertainty about the underlying road map, akin to the Canadian Traveler scenario, each road segment being present with a given probability (this is encoded through a separate, probabilistic, action attempting a segment for the first time). For simplicity, we set that probability to 0.8 throughout.

For pentesting, the general objective is – using *exploits* – to compromise computers in a network, one after another, until specific targets are reached (or no action is available). We modified the POMDP generator by Sarraute, Buffet, and Hoffmann (2012), which itself is based on a test scenario used at Core Security (`http://www.coresecurity.com/`) to output PPDDL encodings of Hoffmann's (2015) *attack-asset MDP* pentesting models. In these models, the network configura-

12. To make the benchmarks feasible for optimal probabilistic planning, we had to reduce their size parameters (number of locations etc). We scaled all parameters with the same number < 1, chosen to get instances at the borderline of feasibility for VI.

tion is known and fixed, and each exploit is callable once and succeeds (or fails) with some probability. The generator uses a network consisting of an exposed part, a sensitive part, and a user part. It allows to scale the numbers H of hosts and E of exploits. Sarraute et al.'s POMDP model and solver (SARSOP, see Kurniawati, Hsu, & Lee, 2008, which does not guarantee optimality) scale to $H = 6, E = 10$.[13] For our benchmarks, we fixed $H = E$ for simplicity (and to obtain a number of instances similar to the other benchmark domains). We scaled the instances from $6 \dots\ 20$ without budget limit, and from $10 \dots\ 24$ with budget limit.

From each of the above benchmark tasks Π (except the pentesting ones for which we already generated a separate limited-budget version anyway), we obtained several limited-budget benchmarks, as follows. We set outcome costs to 1 where not otherwise specified. We determined the minimum budget, $b_{\min}$, required to achieve non-0 goal probability. For the resource-constrained benchmarks, $b_{\min}$ is determined by the generator itself, as the minimum amount of resource required to reach the goal in the deterministic domain version. For all other benchmarks, we ran FD with A* and LM-cut on the all-outcomes determinization of Π. If this failed, we skipped Π, otherwise we read $b_{\min}$ off the cost of the optimal plan and created several limited-budget tasks $\Pi[C]$, differing in their *constrainedness level* C. Namely, following Nakhost et al. (2012), we set the global budget b in $\Pi[C]$ to $b := C * b_{\min}$, so that C is the factor by which the available budget exceeds the minimum needed (to be able to reach the goal at all). We let C range in $\{1.0, 1.2, \dots, 2.0\}$.

For AtleastProb, we let θ range in $\{0.1, 0.2, \dots, 1.0\}$ ($\theta = 0$ is pointless). For ApproxProb, we let δ range in $\{0.0, 0.1, \dots, 0.9\}$ ($\delta = 1$ is pointless). On cyclic problems, the convergence parameter ϵ was set to 0.00005 (the same value as used by Kolobov et al., 2011). All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time/memory cut-offs of 30 minutes/4 GB.

7.2 Acyclic Planning

We consider first acyclic planning. This pertains to all budget-limited benchmarks, to pentesting with and without budget limit, as well as to IPPC TriangleTireworld (moves can be made in only one direction so the state space is acyclic). We consider the 3 objectives MaxProb, AtLeastProb, and ApproxProb. We run all 16 search algorithm variants (VI, AO*, LRTDP, 5 DFHS variants, with subsets of bounds as applicable), each with up to 5 node selection strategies as explained. For dead-end pruning, we run LM-cut, as well as merge-and-shrink *(M&S)* with the state-of-the-art shrinking strategies based on bisimulation and an abstraction-size bound N; we show data for $N = \infty$ and $N = 100k$ (we also tried $N \in \{10k, 50k, 200k\}$ which resulted in similar behavior). We also run variants without dead-end pruning. We use the deterministic-bisimulation (DB) reduced state space only for VI: once (and if) a bisimulation is successfully computed, the block-state MDP is easily solved by that simplest algorithm. Given DB, we do not require any dead-end pruning because all dead-ends are already removed from the reduced state space.

Overall, this yields 577 different possible algorithm configurations. We do not actually test all these configurations, of course, as not all of them are interesting, or needed to make the essential observations. We instead organize our experiment in terms of three parts (1)–(3), each focusing on a particular issue of interest. Consider Table 2, which gives an overview of the configurations considered in each experiment. The design of the experiments is as follows:

13. For modeling/solving the entire network, that is. With their domain-dependent decomposition algorithm "4AL", trading accuracy for performance, Sarraute et al. scale up much further.

Experiment	Search Algorithm	Pruning	Node selection	# Configs
(1) MaxProb search & pruning	VI, $\text{AO}^*\vert_{\text{L}}$, $\text{AO}^*\vert_{\text{U}}$, $\text{LRTDP}\vert_{\text{U}}$, $\text{DFHS}\vert_{\text{U}}$ (5), VI on DB	ALL (4)	default	37
(2) AtLeastProb & ApproxProb parameters	VI, $\text{AO}^*\vert_{\text{L}}$, $\text{AO}^*\vert_{\text{U}}$, $\text{AO}^*\vert_{\text{LU}}$, $\text{LRTDP}\vert_{\text{U}}$, $\text{LRTDP}\vert_{\text{LU}}$, $\text{HDP}\vert_{\text{U}}$, $\text{HDP}\vert_{\text{LU}}$, VI on DB	LM-cut	default	18
(3) AtLeastProb & ApproxProb node selection	VI, $\text{AO}^*\vert_{\text{L}}$, $\text{AO}^*\vert_{\text{U}}$, $\text{AO}^*\vert_{\text{LU}}$, $\text{LRTDP}\vert_{\text{U}}$, $\text{LRTDP}\vert_{\text{LU}}$, $\text{HDP}\vert_{\text{U}}$, $\text{HDP}\vert_{\text{LU}}$, VI on DB	LM-cut	ALL (1, 4, 4, 5, 3, 4, 3, 4, and 1 respectively)	58

Table 2: Overview of algorithms tested on acyclic problems, Section 7.2. Numbers in brackets give the number of options where that number is not obvious. In (2) and (3), note that the total number of configurations gets multiplied by 2 because AtLeastProb vs. ApproxProb result in different algorithm configurations (using different termination criteria). HDP is the $\text{DFHS}_{\mathit{Lab}}^{\mathit{FwdCons}}$ member of our DFHS family, corresponding to Bonet and Geffner's (2003a) HDP algorithm.

(1) We first evaluate different search algorithms and dead-end pruning methods on MaxProb, fixing the node selection strategy to default.

    We omit here all $\text{X}\vert_{\text{LU}}$ variants, because, as explained earlier, for MaxProb heuristic search, maintaining V^L is redundant (early termination is dominated by regular termination).

    Using the default node selection strategy makes sense here because node selection strategies are relevant only for anytime performance, i.e., early termination. This plays a minor role in MaxProb, whose only early termination possibility is the exceptional case where the initial state lower bound becomes $V^L(I) = 1$.

(2) We next fix the best-performing dead-end pruning method, and analyze search algorithm performance in AtLeastProb and ApproxProb as a function of the parameter θ respectively δ.

    We again fix the node selection strategy to default here, leaving their examination to experiment (3).

(3) We finally let the node selection strategies range, keeping otherwise the setting of experiment (2).

We will conclude our discussion with (4) additional data illustrating typical anytime behavior. Each part of the experiment is described in a separate sub-section in what follows.

7.2.1 (1) SEARCH ALGORITHMS & PRUNING METHODS IN MAXPROB

Table 3 shows coverage data, i.e., the number of benchmark tasks for which MaxProb was solved within the given time/memory limits.

Of the pruning methods, LM-cut clearly stands out. For every search algorithm, it yields the by far best overall coverage. M&S has substantial advantages only in RectangleTireworld and NoMystery-b. Note that, for $N = \infty$, overall coverage is worse than for using no pruning at all. This is due to the prohibitive overhead, in some domains, of computing a bisimulation on the determinized state space. And, having invested this effort, it pays off more to use the bisimulation

Domain	#	$\text{DFHS}_{VI}\|_\text{U}$				$\text{DFHS}^{Fwd}_{VI}\|_\text{U}$				$\text{DFHS}^{FwdCons}_{VI}\|_\text{U}$				$\text{DFHS}^{Fwd}_{Lab}\|_\text{U}$				$\text{HDP}\|_\text{U}$			
		–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞
IPPC Benchmarks																					
TriaTire	10	9	**10**	**10**	**10**	9	8	8	8	**10**	**10**	**10**	**10**	9	8	8	8	**10**	**10**	**10**	**10**
IPPC Benchmarks with Budget Limit																					
Blocksw-b	66	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24
Boxworl-b	18	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0
Drive-b	90	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52
Elevator-b	90	78	**86**	79	33	79	**86**	79	33	78	**86**	79	33	79	**86**	79	33	78	**86**	79	33
ExpBloc-b	84	37	60	39	37	37	60	39	37	36	**66**	39	37	37	60	39	37	36	**66**	39	37
Random-b	60	36	**44**	36	33	36	**44**	36	33	36	**44**	36	33	36	**44**	36	33	36	**44**	36	33
RecTire-b	36	28	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**
Tirewor-b	90	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**
TriaTire-b	60	46	**55**	**55**	**55**	46	**55**	**55**	54	46	**55**	**55**	**55**	46	**55**	**55**	54	46	**55**	**55**	**55**
Zenotra-b	36	15	17	17	**18**	15	17	17	**18**	12	15	14	15	15	17	17	**18**	14	16	16	16
Probabilistic Resource-Constrained Benchmarks with Budget Limit																					
NoMystery-b	60	12	40	**50**	**50**	12	41	**50**	**50**	12	42	**50**	**50**	12	41	**50**	**50**	12	42	**50**	**50**
Rovers-b	60	25	46	36	46	25	46	36	46	25	46	36	**47**	25	46	36	46	25	46	36	**47**
TPP-b	60	19	38	27	25	19	38	27	25	20	**39**	27	25	19	38	27	25	20	**39**	27	25
Pentesting Benchmarks																					
Pentest-b	90	57	**63**	62	37	57	**63**	62	37	57	**63**	62	37	57	**63**	62	37	57	**63**	62	37
Pentest	15	**9**	**9**	**9**	8	**9**	**9**	**9**	8	8	8	8	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8
$\sum$	925	575	710	660	554	578	709	658	551	574	716	656	552	578	709	658	551	577	**718**	659	553

Domain	#	VI				$\text{AO}^*\|_\text{L}$				$\text{AO}^*\|_\text{U}$				$\text{LRTDP}\|_\text{U}$				$\text{HDP}\|_\text{U}$				VI on DB
		–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	–	LM	M&S N	M&S ∞	
IPPC Benchmarks																						
TriaTire	10	4	4	4	4	4	4	4	4	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
IPPC Benchmarks with Budget Limit																						
Blocksw-b	66	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24	24	**28**	24	24	24
Boxworl-b	18	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0	0	**3**	0	0	0
Drive-b	90	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52	**90**	**90**	**90**	52	52
Elevator-b	90	71	82	72	33	74	84	76	33	65	77	67	33	79	**86**	79	33	78	**86**	79	33	33
ExpBloc-b	84	32	46	38	37	32	46	38	37	39	57	39	37	38	65	39	37	36	**66**	39	37	37
Random-b	60	27	33	35	33	39	34	36	33	35	**44**	36	33	36	**44**	36	33	36	**44**	36	33	33
RecTire-b	36	30	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**	30	31	**36**	**36**	**36**
Tirewor-b	90	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**
TriaTire-b	60	45	52	52	52	45	52	52	52	46	55	55	55	47	57	57	57	46	55	55	55	**60**
Zenotra-b	36	15	16	16	**18**	15	16	16	**18**	14	16	16	17	15	17	16	17	14	16	16	16	17
Probabilistic Resource-Constrained Benchmarks with Budget Limit																						
NoMystery-b	60	11	37	43	44	11	36	42	43	12	39	47	47	12	41	50	50	12	42	50	50	**51**
Rovers-b	60	23	39	31	40	23	38	31	40	23	44	33	45	25	46	35	46	25	46	36	47	**50**
TPP-b	60	18	35	25	25	16	35	24	24	15	37	26	22	19	38	27	25	20	**39**	27	25	26
Pentesting Benchmarks																						
Pentest-b	90	57	**63**	62	37	57	**63**	62	37	57	**63**	**63**	37	57	**63**	**63**	37	57	**63**	62	37	37
Pentest	15	**9**	**9**	**9**	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8	8
$\sum$	925	546	658	627	533	559	659	630	531	559	693	641	546	581	**718**	661	555	577	**718**	659	553	564

Table 3: Acyclic planning. MaxProb coverage (number of tasks solved within time & memory limits). Best values, within each table, in **boldface**. Top: DFHS variants (recall that HDP is the $\text{DFHS}^{FwdCons}_{Lab}$ member of our DFHS family; DFHS_{VI} is ILAO*). Bottom: remaining search algorithms, including also the overall best DFHS variant. Domains "-b" modified with budget limit. "#": number of instances. "–": no pruning; else pruning, against remaining budget on "-b" domains, based on $h = \infty$ on other domains. "LM": LM-cut; "M&S": merge-and-shrink, "N" size bound $N = 100k$, "∞" no size bound. "VI on DB": VI run on reduced (deterministic-bisimulated) state space. Default node selection.

as a reduced MDP state space ("VI on DB"), rather than only for dead-end pruning. An extreme example of the latter is TriangleTireworld. Far beyond the standard benchmarks in Table 3 (triangle-side length 20), VI on DB scales to side length 74 in both the original domain and the limited-budget version. For comparison, the hitherto best solver by far was Prob-PRP (Camacho et al., 2016), which scales to side length 70 on the original domain, and is optimal only for goal probability 1, i.e., in the presence of strong cyclic plans – which holds for the original domain but not for the limited-budget version. (We could not actually run Prob-PRP on the limited-budget domain version, as Prob-PRP does not natively support a budget, and hard-coding the budget into PPDDL resulted in encodings too large to pre-process.)

Comparing the different DFHS$|_{\mathrm{U}}$ variants, there is no configuration that clearly stands out. Overall, they all perform equally well, though the $^{\mathit{FwdCons}}$ variants (cutting off the exploration at ϵ inconsistent states rather than absorbing states) have a slight edge. This difference mainly comes from TriangleTireworld, ExplodingBlocks, and TPP-b, where the $^{\mathit{FwdCons}}$ configurations solve more instances, while in Zenotravel-b the $^{\mathit{FwdCons}}$ configurations perform slightly worse than their counterparts. The termination parameter (`VI` vs. `Label`) has almost no effect on coverage. Due to the similarity of the DFHS configurations, and because DFHS$^{\mathit{FwdCons}}_{\mathit{Lab}}$ gives the best coverage results, we will use DFHS$^{\mathit{FwdCons}}_{\mathit{Lab}}$ as the representative of the DFHS family in the remaining discussion. As DFHS$^{\mathit{FwdCons}}_{\mathit{Lab}}$ corresponds to HDP, for simplicity we will from now on refer to it by that name.

AO$^*|_{\mathrm{L}}$ is better than VI only in case of early termination on $V^L = 1$, when a full-certainty policy is found before visiting the entire state space. This happens very rarely here, and AO$^*|_{\mathrm{L}}$ is dominated by VI (this changes for AtLeastProb, see Figures 5a and 7 below). All failures of VI are due to memory or runtime exhaustion while building the reachable state space. LRTDP$|_{\mathrm{U}}$ clearly outperforms AO$^*|_{\mathrm{U}}$, presumably because it tends to find absorbing states more quickly. LRTDP$|_{\mathrm{U}}$ and HDP$|_{\mathrm{U}}$ are about on par; with LM-cut they solve the exact same number of instances (though not exactly the same instances), and otherwise HDP$|_{\mathrm{U}}$ solves slightly fewer tasks than LRTDP$|_{\mathrm{U}}$.

To gauge the efficiency of heuristic search vs. blind search on MaxProb, compare LRTDP$|_{\mathrm{U}}$ vs. VI in Table 3. Contrary to the intuition that a good initial goal probability estimator is required for heuristic search to be useful, LRTDP$|_{\mathrm{U}}$ is clearly superior. Its advantage does grow with the quality of the initialization; LM-cut yields the largest coverage increase by far. However, even without dead-end pruning, i.e., with the trivial initialization of V^U, LRTDP$|_{\mathrm{U}}$ dominates VI throughout, and improves coverage in 8 of the 16 domains.

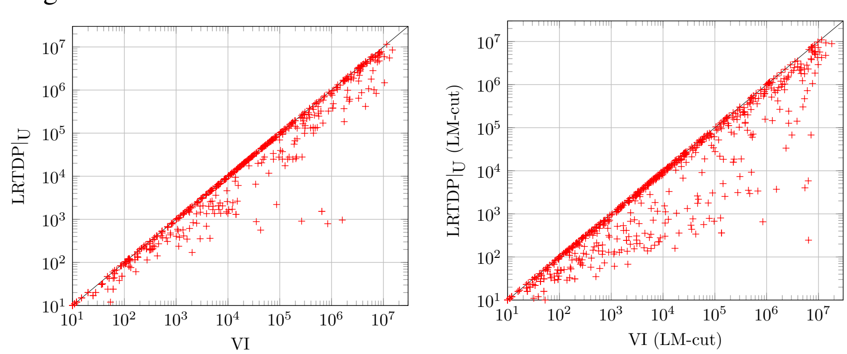


Figure 4: Acyclic planning. Number of states visited, for VI (x) vs. LRTDP$|_{\mathrm{U}}$ (y), with no pruning (left) respectively LM-cut pruning (right). Default node selection.

We next shed additional light on this by comparing search space sizes and runtime values. Tables 4 and 5 provide aggregate data, Figure 4 gives a scatter plot for the canonical comparison

		VI				$\text{AO}^*\|_\text{U}$				$\text{LRTDP}\|_\text{U}$				$\text{HDP}\|_\text{U}$			
		–	LM	M&S		–	LM	M&S		–	LM	M&S		–	LM	M&S	
Domain	#			N	∞			N	∞			N	∞			N	∞
IPPC Benchmarks																	
TriaTire	1	843.1	843.1	843.1	843.1	0.2	0.2	0.2	0.2	0.8	0.7	0.7	0.7	2.2	1.8	1.8	1.8
ONLY-H	4					0.4	0.4	0.4	0.4	3.5	1.7	1.7	1.7	160.3	835.4	835.4	835.4
IPPC Benchmarks with Budget Limit																	
Blocksw-b	18	12.7	5.8	2.8	2.8	12	5.2	2.5	2.5	12.3	5.3	2.5	2.5	11.5	4.8	2.3	2.3
Drive-b	20	4.2	2.4	1.8	1.2	4.2	2.2	1.6	1	4	2.1	1.5	1	4	2.1	1.5	0.9
Elevator-b	12	12.8	3.9	7.2	3	3.3	0.3	0.4	0.1	3.6	0.3	0.4	0.1	3.4	0.3	0.4	0.1
ExpBloc-b	18	1.1K	41.9	92.1	33.2	112.2	1.2	12.2	0.8	117.8	1.4	12.5	1.2	150.7	1.2	13.6	0.8
NON-TRIVIAL	7	4.1K	213.2	859.9	179	603.1	3.6	133.6	2.6	588	4	106.8	3.2	780.4	3.4	125.6	2.2
ONLY-H	3					2.6K	1.5	17.2	1.1	3.1K	2.1	21.3	1.6	5.2K	1.9	23.4	1.1
Random-b	21	4.5	1.7	1.7	1.7	1.9	0.8	0.8	0.8	2	0.8	0.8	0.8	1.9	0.8	0.8	0.8
NON-TRIVIAL	4	1.0K	130	127.4	127.4	42.6	6.4	6.4	6.4	43.7	6.4	6.4	6.4	34.5	5.4	5.4	5.4
ONLY-H	2					2.7K	15.9	2.2	2.2	2.9K	15.9	2.2	2.2	2.9K	15.9	2.2	2.2
RecTire-b	18	50.6	5.6	1.5	1.5	49.8	5.1	1.2	1.2	50.4	5.1	1.3	1.3	50.4	5	1.2	1.2
NON-TRIVIAL	12	81.9	8.9	2.4	2.4	81.5	8.2	1.9	1.9	81.6	8.3	2	2	81.5	8.1	1.9	1.9
TriaTire-b	17	1.4K	6.4	6.4	6.4	898.6	3	3	3	896.2	2.9	2.9	2.9	954	3.4	3.4	3.4
NON-TRIVIAL	6	4.1K	229.4	229.4	229.4	1.8K	52.5	52.5	52.5	1.6K	44.5	44.5	44.5	1.8K	67.4	67.4	67.4
ONLY-H	1					7.4K	610.3	610.3	610.3	4.6K	303.3	303.3	303.3	6.2K	634.8	634.8	634.8
Zenotra-b	14	491.5	30.2	35.8	18.2	491.3	29.9	35.2	17.9	288.2	23.6	27.1	14.1	285.1	23.6	27.4	14.2
NON-TRIVIAL	10	967.4	104.4	164.6	64	967.1	102.9	161.1	62.3	478.1	75.3	114.9	45.8	468.6	75.3	115.9	46.3
Probabilistic Resource-Constrained Benchmarks with Budget Limit																	
NoMystery-b	11	2.8K	6.9	0.5	0.5	2.6K	6.6	0.4	0.4	2.6K	6.4	0.4	0.4	2.7K	6.5	0.4	0.4
ONLY-H	1					12.4K	122.4	14.1	14.1	12.7K	122.3	16.4	16.4	12.7K	117.3	14.2	14.2
Rovers-b	21	1.1K	51.8	91.5	22.6	702.9	36.1	58.6	12.4	873.7	38.5	70.8	14.3	782.7	35.8	63.2	12.4
NON-TRIVIAL	13	2.2K	290.1	512.6	137.6	1.1K	176	281.4	65.9	1.6K	190	366.2	76.3	1.3K	173.8	318.4	65.9
ONLY-H	2					4.7K	287.1	709.7	205.2	8.3K	338.3	1.0K	242.1	5.9K	265.5	741	189.5
TPP-b	9	1.1K	49.6	265.4	10.9	660.9	33	183.3	5.7	897.2	38.5	220.7	7.6	765.4	31.3	188.1	5.6
NON-TRIVIAL	5	3.0K	178.7	894.6	36.6	1.5K	100.4	549.5	14.3	2.1K	120.1	701.9	21.5	1.8K	91.3	561.8	14
Pentesting Benchmarks																	
Pentest-b	28	19.7	6.3	7.8	6.3	19.5	6.3	7.7	6.3	19.7	6.2	7.7	6.2	19.7	6.2	7.7	6.2
NON-TRIVIAL	5	238.1	165.1	169.2	165.1	237.2	165.1	169	165.1	238.1	165.1	169.1	165.1	238.1	165.1	169.1	165.1
Pentest	3	74.3	66.3	66.4	66.3	74.3	66.3	66.4	66.3	74.3	66.3	66.4	66.3	74.3	66.3	66.4	66.3
NON-TRIVIAL	1	194.3	173.4	173.8	173.4	194.3	173.4	173.8	173.4	194.3	173.4	173.8	173.4	194.3	173.4	173.8	173.4

Table 4: Acyclic planning. MaxProb geometric mean search space size (number of states visited) in multiples of 1000. "#" gives the size of the instance basis, namely those instances solved by all shown configurations, skipping instances solved in under 1 second by all configurations. "NON-TRIVIAL" uses only those instances not solved by VI in < 1 second. "ONLY-H" uses those instances commonly solved by $\text{AO}^*|_\text{U}$, $\text{LRTDP}|_\text{U}$, and $\text{HDP}|_\text{U}$, but not solved by VI. Rows with empty instance basis are skipped. Default node selection.

between VI and $\text{LRTDP}|_\text{U}$. Data for $\text{AO}^*|_\text{L}$ is not shown as its coverage is dominated by VI (cf. Table 3), and the same goes for its runtime and search space. We include the "NON-TRIVIAL" rows in the tables to show behavior on the more interesting instances, where the averages are not skewed by the many very small instances in most domains. We include the "ONLY-H" rows to elucidate the behavior on the most challenging instances beyond reach of VI.

A clear message from Table 4 and Figure 4 is that the heuristic search algorithms, apart from a few exceptions, visit much fewer states than VI does, even with trivial upper bound initialization where search spaces are reduced in all domains except RectangleTireworld and Pentest. For instance, using $\text{LRTDP}|_\text{U}$ instead of VI results in a gain of around 1 order of magnitude in many instances, and larger gains (up to 3 orders of magnitude) also occur in rare cases. By giving the heuristic search algorithms additional information through earlier dead end detection, the differences become even larger.

		VI				AO*$\|_U$				LRTDP$\|_U$				HDP$\|_U$			
		–	LM	M&S		–	LM	M&S		–	LM	M&S		–	LM	M&S	
Domain	#			N	∞			N	∞			N	∞			N	∞
IPPC Benchmarks																	
TriaTire	1	3.5	7.4	4.1	4	0	0	0.1	0.1	0	0	0	0	0	0	0.1	0.1
ONLY-H	4					0.1	0.1	0.5	0.5	0.1	0.3	0.6	0.6	1.4	23.9	12.1	12.1
IPPC Benchmarks with Budget Limit																	
Blocksw-b	18	0.1	0.6	2.5	2.3	1.8	0.8	3	2.8	0.2	0.6	2.3	2.4	0.2	0.5	2	2.1
Drive-b	20	0	0.2	6.9	14	0.1	0.2	8	15.4	0	0.2	7.1	14.2	0	0.2	6.1	12.3
Elevator-b	12	0.1	0.1	1.8	4.1	0	0	2.2	4.5	0	0	1.8	4.1	0	0	1.7	3.8
ExpBloc-b	18	6	1	15.5	7.5	1.6	0	16	8	0.8	0.1	14.2	7.1	0.9	0	13.4	6.6
NON-TRIVIAL	7	25.3	4.8	36.2	45.7	11.5	0.1	33	45.9	4	0.1	29.7	42.1	5	0.1	27.2	39.6
ONLY-H	3					29.3	0.1	40.9	40.4	21.4	0.1	30.7	34.9	35.1	0.1	30.2	32.5
Random-b	21	0.5	0.6	4.8	4.8	0.3	0.3	5.2	5.2	0.3	0.3	4.7	4.8	0.2	0.3	4.4	4.4
NON-TRIVIAL	4	13.9	10.1	39.2	43.2	3	0.9	44.4	49.9	1.4	0.8	36.3	43.2	0.8	0.7	35.3	38.2
ONLY-H	2					27.8	11.3	38.1	42.9	30.3	11.1	35.4	37.4	29.8	10.8	34.8	35.2
RecTire-b	18	9	19.4	1.2	1.2	73.6	20.2	1.3	1.3	43.1	17.6	1.3	1.3	131.2	16.4	1.2	1.2
NON-TRIVIAL	12	20.4	57.3	2.3	2.3	178.9	61.8	2.4	2.4	106.8	51.4	2.3	2.3	330.1	46.5	2.3	2.3
TriaTire-b	17	10.5	0.5	0.6	0.6	14.5	0.4	0.5	0.5	9.5	0.3	0.4	0.4	8.4	0.4	0.4	0.4
NON-TRIVIAL	6	27.7	5.9	3.2	3.3	31.3	2.4	2.1	2	14.7	2	1.7	1.6	13.7	2.4	1.8	1.7
ONLY-H	1					153.2	25.4	13.6	13.6	41.5	11.9	5.1	4.4	42	18.2	6.9	7.1
Zenotra-b	14	2.7	4.9	15	9	56	5.7	18.9	11.8	13	4.3	15.9	9.2	52.5	5	16.8	9.3
NON-TRIVIAL	10	5.6	16.5	27	13.5	163.4	19.3	37.2	18.7	25.3	13.6	30.1	14.5	118.3	16.8	35.2	15.1
Probabilistic Resource-Constrained Benchmarks with Budget Limit																	
NoMystery-b	11	15.6	0.4	0.3	0.3	242.6	0.4	0.3	0.3	27.8	0.4	0.3	0.3	26.2	0.4	0.3	0.3
ONLY-H	1					1623.4	8.9	0.6	0.6	158.8	7.8	0.5	0.4	137.2	7.3	0.4	0.4
Rovers-b	21	9.7	2.3	11.8	17	96.2	2.3	16.5	21.7	12.8	2	11.6	16.1	10.8	1.8	9.9	14.9
NON-TRIVIAL	13	20.9	12.9	20.2	33.7	236.6	12	33.3	45.1	24.9	9.7	19.5	30.3	19.2	8.8	15.9	29.1
ONLY-H	2					751.2	19.2	76.9	151.4	127.6	18.6	44.8	126.9	85.4	14.8	34.3	105.1
TPP-b	9	8.5	1.6	14.9	69.8	63.2	1.3	24.6	70.2	12.7	1.3	16	69.1	9.6	1.1	14.3	65.1
NON-TRIVIAL	5	22.4	5.7	18.5	76.2	203.5	4.5	37.3	78.2	31.3	4.2	20.1	76.3	23.2	3.2	17.7	72.4
Pentesting Benchmarks																	
Pentest-b	28	0	0	6.6	16.5	0.5	0	8.2	19.8	0	0	7.3	18.2	0.3	0	6.5	16.6
NON-TRIVIAL	5	3.2	2.2	10.1	92.7	16	4.5	15	108.9	8.5	5.2	15.2	107.6	6.4	4.4	12.7	100
Pentest	3	0.9	0.7	3.2	4.4	5.9	2.3	5.7	6.5	3.8	3	5.8	6.3	2.4	2.7	5.1	6.1
NON-TRIVIAL	1	2.7	2	6.5	17.2	23	8.1	20.6	23.8	15	10.4	16.6	24.4	9.3	10.6	15.2	24.4

Table 5: Acyclic planning. MaxProb geometric mean runtime (in CPU seconds). Same setup and presentation as in Table 4.

As previously hinted, these observations have not been made in this clarity before. While Kolobov et al. (2011) also report LRTDP to beat VI on MaxProb, they consider only a single domain; they do not experiment with trivially initialized V^U; and they do not use dead-end pruning in VI, so that LRTDP already benefits from a smaller state space, and the impact of heuristic search remains unclear.

Even though the search space of the heuristic search algorithms is in many cases only a small fraction of the whole (dead-end pruned) state space, this is not necessarily reflected in runtime. On those instances solved by VI, it is typically fast, often faster than heuristic search and rarely outperformed significantly. This is despite having larger search spaces, i.e., heuristic search does visit less states but suffers from having to do more updates on these (recall that VI here updates each visited state exactly once). Significant runtime advantages over VI (in the "NON-TRVIAL" rows) are obtained by heuristic search only in ExplodingBlocks–b, Random–b, and TriangleTireworld–b.

Comparing the heuristic search algorithms, the conclusions are more fine-grained but overall similar to what we concluded from coverage above. LRTDP$|_U$ dominates AO*$|_U$ almost throughout. Note that, even though the search space size of AO*$|_U$ and LRTDP$|_U$ almost always is similar, AO*$|_U$ requires a lot more time than LRTDP$|_U$. This is because it performs more updates. Across the non-trivial commonly solved instances in the tables, the geometric mean of the number of updates done in AO$^*|_\mathrm{U}$ is about 4 times higher than that in LRTDP$|_\mathrm{U}$. LRTDP$|_\mathrm{U}$ and HDP$|_\mathrm{U}$ with non-trivial value initialization give very similar results, not only in terms of coverage, but also in terms of runtime and search space size. LRTDP$|_\mathrm{U}$ is, however, more effective in some of the domains (e.g., RectangleTireworld and Zenotravel) if no additional dead-end detection method is used. On the other hand, HDP$|_\mathrm{U}$ has a slight edge in the probabilistic resource-constrained domains. One notable case where LRTDP$|_\mathrm{U}$ consistently outperforms HDP$|_\mathrm{U}$ is TriangleTireworld.

The impact of dead-end pruning on VI is typically moderate. The gains for heuristic search are much more pronounced, thanks to the stronger heuristic function initialization. Especially AO$^*|_\mathrm{U}$ benefits a lot. LRTDP$|_\mathrm{U}$ and HDP$|_\mathrm{U}$ benefit as well, but to a smaller extent, partly because they are already more effective in the first place. Comparing across different dead-end pruning methods, although M&S with $N = \infty$ clearly yields the largest search space reductions, and necessarily so as it recognizes *all* dead-ends, the overhead of bisimulation computation outweighs the search space reduction in all but a few cases. In terms of pruning power, M&S with $N = 100k$ and the LM-cut heuristic are overall roughly similar, yet LM-cut has the edge in runtime.

7.2.2 (2) ATLEASTPROB AND APPROXPROB PARAMETER ANALYSIS

We now turn to the weaker objectives, AtLeastProb and ApproxProb. We fix LM-cut for the (almost always most effective) dead-end pruning. We examine the power of early termination for different search algorithms and node selection strategies. This is best viewed as a function of the goal probability threshold θ in AtLeastProb, and of the desired goal probability accuracy δ in ApproxProb. VI forms a baseline independent of θ (δ). Consider Figure 5.

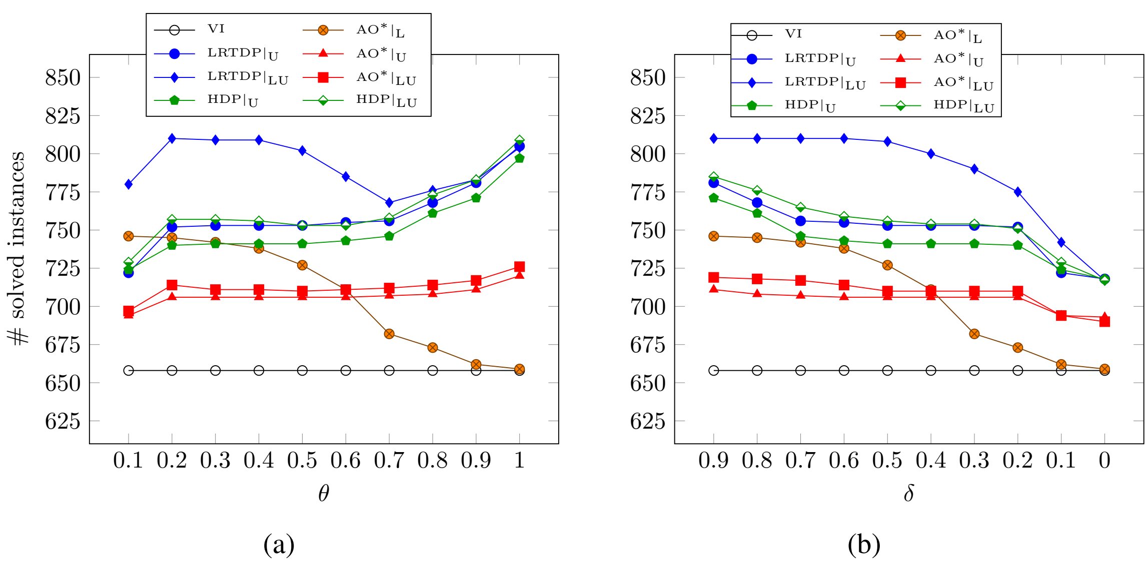


Figure 5: Acyclic planning. Total coverage for AtLeastProb as a function of θ in (a), for ApproxProb as a function of δ in (b). All configurations use default node selection and LM-cut dead-end pruning.

For AtLeastProb (Figure 5a), in the interesting region of benchmark instances not feasible for VI yet sometimes feasible for the other search algorithms, one clear feature is the superiority of LRTDP over both AO* and HDP. As one can see for the smaller values of θ, LRTDP is able to update V^L much more effectively than HDP, resulting in a larger coverage of LRTDP in the region of smaller θ values. AO*$|_\mathrm{L}$ exhibits strikingly strong behavior for small values of θ, approaching (and in one case, surpassing) the performance of LRTDP$|_\mathrm{U}$. Evidently, the depth-first expansion strategy is quite effective for anytime behavior on V^L and thus for termination via $V^L(I) \geq \theta$. It is way more effective than the heuristic search in AO*$|_\mathrm{LU}$. As we shall see below (Figure 7), it is often also more effective than LRTDP. In general, for all algorithms, using V^L is a clear advantage for small θ. For larger θ, maintaining V^L can become a burden, yet V^U is of advantage due to early termination on $V^U(I) < \theta$. Algorithms using both bounds exhibit an easy-hard-easy pattern.

The spike at the left-hand side in Figure 5 (a), i.e., significantly worse performance for $\theta = 0.1$ than for $\theta = 0.2$, is an outlier due to the Pentest domains – without these domains, AO*$|_\mathrm{LU}$, LRTDP$|_\mathrm{LU}$ and HDP$|_\mathrm{LU}$ exhibit a strict easy-hard-easy pattern. This is because, in contrast to typical probabilistic planning scenarios, in penetration testing the goal probability – the chance of a successful attack – are typically small, and indeed this is so in our benchmarks. Searches using an upper bound quickly obtain $V^U(I) < 0.2$, terminating early based on $V^U(I) < \theta$ for $\theta = 0.2$. But it takes a long time to obtain $V^U(I) < 0.1$.

For ApproxProb (Figure 5b), smaller values of δ consistently result in worse performance. We see again the superiority of LRTDP over AO* and HDP, and the (relatively, compared to AO*$|_\mathrm{LU}$) strong behavior of AO*$|_\mathrm{L}$ in δ regions allowing aggressive early termination. Again, the key to LRTDP beating HDP so clearly is due to LRTDP updating V^L much more effectively. HDP$|_\mathrm{LU}$ can only improve on HDP$|_\mathrm{U}$ by a small margin. Nonetheless, we see again the superiority of algorithms using both bounds over those that don't.

7.2.3 (3) NODE SELECTION STRATEGIES

Figure 6 shows different node selection strategies in AtLeastProb (the relative performance of node selection strategies is the same in ApproxProb, so we do not include a separate figure for that).

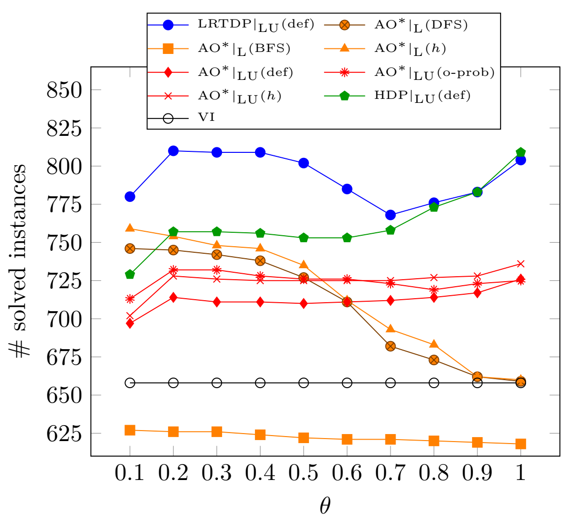
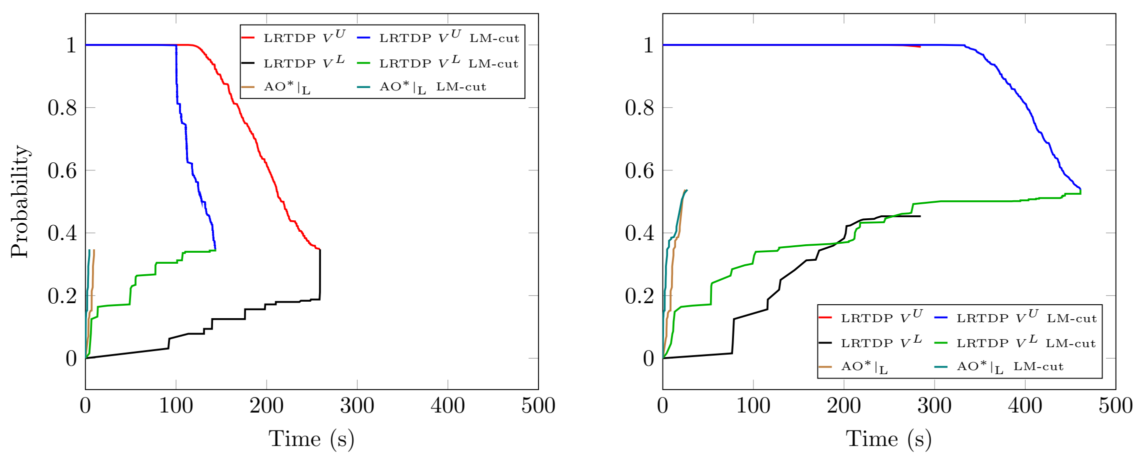


Figure 6: Acyclic planning. Total coverage for AtLeastProb as a function of θ, varying the node selection strategy. All configurations use LM-cut dead-end pruning.

For readability, we show only the most competitive base algorithms, AO*$|_\mathrm{L}$, AO*$|_\mathrm{LU}$, LRTDP$|_\mathrm{LU}$, and HDP$|_\mathrm{LU}$ (as well as the VI baseline). For LRTDP and HDP, we show only default node selection, which consistently works basically as well as the alternatives. For AO*$|_\mathrm{L}$, we see that the depth-first strategy is important (and way beyond breadth-first, which does worse than VI). The h-bias strategy is marginally, but consistently, better than depth-first. For AO$^*|_{\mathrm{LU}}$, both the h-bias and the most-prob-outcome bias are helpful, substantially improving over the default strategy. The h-bias consistently improves a bit on default AO*. The gap-bias and preferred-actions strategies are not shown as they were consistently slightly worse (apparently, the gap-bias leads to a more breadth-first style behavior, while preferred actions mainly cause runtime overhead).

7.2.4 (4) AN ILLUSTRATION OF TYPICAL ANYTIME BEHAVIOR

To conclude our discussion of acyclic planning, Figure 7 exemplifies typical anytime behavior, i.e., the development of the $V^L(I)$ and $V^U(I)$ bounds on the initial state value, as a function of runtime, for LRTDP$|_{\mathrm{LU}}$ and AO$^*|_{\mathrm{L}}$.



(a) (b)

Figure 7: Acyclic planning. Anytime behavior in LRTDP$|_{\mathrm{LU}}$ (V^U and V^L) and AO$^*|_{\mathrm{L}}$ (V^L only), as a function of runtime. Elevators instance 11, without pruning and with LM-cut pruning, for constrainedness level $C = 1.4$ (a) respectively $C = 1.8$ (b). Default node selection.

The benefit of LM-cut pruning is evident. Observe that AO$^*|_{\mathrm{L}}$ is way more effective than LRTDP in quickly improving the lower bound. Indeed, the runs shown here find an optimal policy very quickly. Across the benchmarks solved by both AO$^*|_{\mathrm{L}}$ and LRTDP, omitting those where both took < 1 second, in 56% of cases AO$^*|_{\mathrm{L}}$ finds an optimal policy faster than LRTDP. On (geometric) average, AO$^*|_{\mathrm{L}}$ takes 66% of the time taken by LRTDP for this purpose. On the downside, unless $V^*(I) \geq \theta$, AO$^*|_{\mathrm{L}}$ must explore the entire state space. Its runs in Figure 7 exhaust memory for MaxProb. In summary, heuristic search is much stronger in proving that the maximum goal probability is found, but is often distracting for improving V^L quickly.

As both parts of Figure 7 use the same base instance but with different constrainedness levels C, we can also draw conclusions on the effect of surplus budget. With more budget, more actions can be applied before reaching absorbing states. This adversely affects the upper bound (consistently across our experiments), which takes a much longer time to decrease. The lower bound, on the other hand, often increases more quickly with higher C as it is easier to find goal states.

7.3 Cyclic Planning with FRET

We now consider cyclic planning, pertaining to the standard IPPC benchmarks, and to probabilistic NoMystery, Rovers, TPP without budget (nor resource-) limit. We run only LRTDP and DFHS, as AO* is restricted to acyclic state spaces. We use the two different variants of FRET described earlier: FRET-V^U as per Kolobov et al. (2011), and our new variant FRET-π^U. We consider all 3 objectives, and the same 4 dead-end pruning methods (as LM-cut returns ∞ iff the cheaper heuristic $h^{\max}$ does, we use $h^{\max}$ here). We do not vary node selection strategies because, like we have seen before, in LRTDP and DFHS these do not bring a notable advantage over the default strategy. We use the deterministic-bisimulation (DB) reduced state space with each base algorithm, as some differences do emerge (in difference to the acyclic case) between VI and the other algorithms, which now need to run FRET. Again, given DB we do not require any dead-end pruning.

Overall, this yields 305 different possible algorithm configurations. As before, not all of these are interesting, and we instead organize our experiment in terms of parts focusing on issues of interest. Specifically, we have parts (1) on MaxProb and (2) on AtLeastProb/ApproxProb as before. As node selection strategies are not relevant here, we do not have the previous part (3) considering these. We integrate data illustrating anytime behavior with our discussion of (2). Table 6 gives an overview of tested configurations.

Experiment	FRET variant	Search Algorithm	Pruning	# Configs
(1) MaxProb search & pruning	–, FRET-V^U, FRET-π^U	VI, LRTDP$\|_{\mathrm{U}}$, DFHS (5)	ALL (4), DB	65
(2) AtLeastProb & ApproxProb parameters	–, FRET-V^U, FRET-π^U	VI, LRTDP$\|_{\mathrm{U}}$, LRTDP$\|_{\mathrm{LU}}$, HDP$\|_{\mathrm{U}}$, HDP$\|_{\mathrm{LU}}$	$h^{\max}$	18

Table 6: Overview of algorithms tested on cyclic problems, Section 7.3. Note that VI does not require, and is hence not combined with, FRET; we denote this (not using FRET at all) by "–". In (2), note that the number of configurations gets multiplied by 2 because AtLeastProb vs. ApproxProb result in different algorithm configurations (using different termination criteria). All configurations tested use default node selection.

7.3.1 (1) SEARCH ALGORITHMS & PRUNING METHODS IN MAXPROB

Table 7 shows coverage data. As before, the DFHS family is shown at the top, and the remaining search algorithms, including the most competitive DFHS algorithm which as before is HDP, are shown at the bottom. We do not vary the FRET variant at the top for space reasons, and as, for FRET-V^U, there were no coverage differences at all across DFHS family members.

Similarly as in the acyclic case, the DFHS configurations stopping exploration at ϵ-inconsistent states give slightly better results than those stopping only at absorbing states. The termination parameter has almost no effect on coverage: HDP (i.e., $\mathrm{DFHS}_{Lab}^{FwdCons}$) solves one more task in ExplodingBlocks than $\mathrm{DFHS}_{VI}^{FwdCons}$, but otherwise the coverage is the same. Also akin to the acyclic case, both LRTDP and HDP perform equally well, though now HDP has a slight edge in combination with FRET-π^U.

Running the search on a deterministic-bismulation state space is less effective on the cyclic benchmarks than on the acyclic ones. It gives a clear advantage only in Rovers.

Domain	#	FRET-π^U: DFHS$_{VI}$|$_U$					DFHS$_{VI}^{Fwd}$|$_U$					DFHS$_{VI}^{FwdCons}$|$_U$					DFHS$_{Lab}^{Fwd}$|$_U$					HDP|$_U$				
		–	$h^{\max}$	M&S N	M&S ∞	on BS	–	$h^{\max}$	M&S N	M&S ∞	on BS	–	$h^{\max}$	M&S N	M&S ∞	on BS	–	$h^{\max}$	M&S N	M&S ∞	on BS	–	$h^{\max}$	M&S N	M&S ∞	on BS
IPPC Benchmarks																										
Blocksworld	15	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Boxworld	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Drive	15	4	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6
Elevators	15	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5
ExplodingBlocks	15	5	12	5	4	4	5	12	5	4	4	5	14	5	4	4	5	12	5	4	4	5	**15**	5	4	4
Random	15	**6**	**6**	1	0	0	**6**	**6**	2	0	0	**6**	**6**	1	0	0	**6**	**6**	2	0	0	**6**	**6**	1	0	0
RectangleTireworld	14	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
Tireworld	15	**15**	**15**	**15**	12	11	**15**	**15**	**15**	12	11	**15**	**15**	**15**	12	11	**15**	**15**	**15**	12	11	**15**	**15**	**15**	12	11
Zenotravel	15	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1
Probabilistic Resource-Constrained Benchmarks																										
NoMystery	10	**4**	**4**	**4**	**4**	0	**4**	**4**	**4**	**4**	0	**4**	**4**	**4**	**4**	0	**4**	**4**	**4**	**4**	1	**4**	**4**	**4**	**4**	1
Rovers	10	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**	8	**9**
TPP	10	**8**	**8**	**8**	6	6	**8**	**8**	**8**	6	6	**8**	**8**	**8**	6	6	**8**	**8**	**8**	6	6	**8**	**8**	**8**	6	6
$\sum$	164	87	105	93	64	60	98	105	94	64	60	98	107	93	64	60	98	105	94	64	61	98	**108**	93	64	61

Domain	#	VI					FRET-V^U: LRTDP|$_U$					FRET-V^U: HDP|$_U$					FRET-π^U: LRTDP|$_U$					FRET-π^U: HDP|$_U$				
		–	$h^{\max}$	M&S N	M&S ∞	on DB	–	$h^{\max}$	M&S N	M&S ∞	on DB	–	$h^{\max}$	M&S N	M&S ∞	on DB	–	$h^{\max}$	M&S N	M&S ∞	on DB	–	$h^{\max}$	M&S N	M&S ∞	on DB
IPPC Benchmarks																										
Blocksworld	15	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Boxworld	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Drive	15	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6	**15**	**15**	**15**	6	6
Elevators	15	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5	**15**	**15**	**15**	5	5
ExplodingBlocks	15	4	6	4	4	4	4	6	4	4	4	4	6	4	4	4	5	14	5	4	4	5	**15**	5	4	4
Random	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	0	**6**	**6**	1	0	0
RectangleTireworld	14	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
Tireworld	15	10	10	10	10	11	10	11	10	10	11	10	10	10	10	11	**15**	**15**	**15**	12	11	**15**	**15**	**15**	12	11
Zenotravel	15	**3**	**3**	**3**	1	0	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1	**3**	**3**	**3**	1	1
Probabilistic Resource-Constrained Benchmarks																										
NoMystery	10	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	4	4	4	4	1	4	4	4	4	1
Rovers	10	5	5	5	5	**9**	5	5	5	5	**9**	5	5	5	5	**9**	**9**	**9**	**9**	8	**9**	**9**	**9**	**9**	8	**9**
TPP	10	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	**8**	**8**	**8**	6	6	**8**	**8**	**8**	6	6
$\sum$	164	81	83	81	60	64	81	84	81	60	65	81	83	81	60	65	96	105	92	64	61	98	**108**	93	64	61

Table 7: Cyclic planning. MaxProb coverage. Best values, within each table, in **boldface**. FRET-V^U is as per Kolobov et al. (2011), FRET-π^U is our modified version. Top: DFHS variants (recall that HDP is the DFHS$_{Lab}^{FwdCons}$ member of our DFHS family; DFHS$_{VI}$ is ILAO*); showing only the dominating FRET version, FRET-π^U. Bottom: remaining search algorithms, varying the FRET version, and including also the overall best DFHS variant. Dead-end pruning variants: "–" none, else based on heuristic value ∞, for $h^{\max}$ respectively merge-and-shrink ("N" size bound $N = 100k$, "∞" no size bound). "on DB": run on reduced (deterministic-bisimulated) state space. Default node selection.

The most striking result here by far is that FRET-π^U outperforms both VI and FRET-V^U substantially. Note that, in all domains except ExplodingBlocks and Rovers, the advantage over VI is obtained even without dead-end pruning, i.e., for trivial initialization of V^U. This strongly confirms the power of heuristic search even in the absence of good admissible goal probability estimators.

As before, we shed additional light on the coverage results through search space size and runtime data. Figure 8 compares the search space sizes for VI vs. FRET-π^U. The non-trivial initialization using $h^{\max}$ is useful, but gains of up to 3 orders of magnitude are possible even without it.

Table 8 provides aggregate search space size and runtime data. No data is shown for the configuration using FRET-V^U with HDP, as that data is almost identical to that of FRET-V^U with LRTDP:

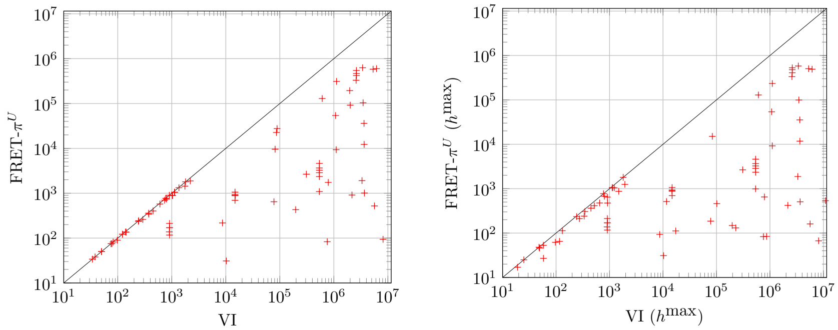


Figure 8: Cyclic planning. Number of states visited, for VI (x) vs. FRET-π^U using LRTDP$|_U$ (y), with no pruning (left) respectively $h^{\max}$ pruning (right).

		VI				FRET-V^U LRTDP$\vert_U$				FRET-π^U LRTDP$\vert_U$				FRET-π^U HDP$\vert_U$			
		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S	
Domain	#			N	∞			N	∞			N	∞			N	∞
IPPC Benchmarks																	
Blocksworld	4	0	0	2.8	2.7	0	0.1	2.7	2.8	0.1	0.1	3	2.8	0.1	0.1	3.1	2.9
Drive	1	0	0	6	42.8	0	0	5.8	33.1	0	0	6.1	42.8	0	0	7.2	33.5
Elevators	5	0	0	2.2	1.7	0	0	2.2	1.8	0	0	2.2	1.9	0	0	2.2	1.8
ExplodingBlocks	4	2.6	0.7	19.3	18.1	15.9	0.4	31.5	17.7	15.7	0	28.8	17.5	3.5	0	18.9	18.3
NON-TRIVIAL	2	14.2	2.7	45.1	110.4	82.2	0.8	112.4	102.6	72.9	0	104.1	110.1	14.3	0	44.4	107.1
RectangleTireworld	6	3.7	4	4.7	4.7	3.9	4	4.7	4.7	19.8	4.1	4.7	4.7	20.4	4.1	4.7	4.7
NON-TRIVIAL	4	7.2	7.9	9.1	9	7.6	7.9	9.1	9	53	8.1	9.1	9.2	55.5	8.1	9.2	9.1
Tireworld	8	7.3	10.9	14	35	60	55.3	60.2	86.4	0	0	3.8	24.7	0	0	3.8	24.9
NON-TRIVIAL	7	11	16.5	19.5	55.7	92.1	84.5	89.8	136	0	0	4.7	39.8	0	0	4.6	39.9
Zenotravel	1	55.7	49.3	96.3	283.8	7	12.6	54.3	241.1	0.2	0.2	43.5	227.4	0.2	0.2	51	233.9
Probabilistic Resource-Constrained Benchmarks																	
NoMystery	4	21.5	29.8	29.1	69.4	133.9	127	141.4	166.7	627.3	582.4	676.4	618.7	632.1	580.4	634.2	628.5
Rovers	5	33.1	40.2	39.1	42.7	439.8	420.3	435.2	425.1	1.8	1.3	6.2	8.3	1.8	1.3	6.1	8.3
TPP	6	11.8	14.4	20.1	44.8	140.1	125.8	136.6	156.3	32.9	18.3	63.2	71.3	32.9	18.5	64.1	70.3
NON-TRIVIAL	5	21.6	26.2	31.6	83.5	259.1	241	253.2	299.1	52.5	31.8	118.3	138.8	52.9	32.3	120.1	137.1
IPPC Benchmarks																	
Blocksworld	4	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
Drive	1	0.3	0.3	0.3	0.3	0.3	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.3	0.2	0.2	0.2
Elevators	5	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
ExplodingBlocks	4	408.5	46.5	252.3	39.9	408.5	20.1	242.2	16.9	44.3	0.2	14	0.1	44.4	0.2	14	0.1
NON-TRIVIAL	2	2.0K	152.6	2.0K	142.4	2.0K	34.6	2.0K	32.4	133.6	0.2	133.6	0.2	133.7	0.2	133.7	0.2
RectangleTireworld	6	0.7	0.2	0	0	0.7	0.2	0	0	0.7	0.2	0	0	0.7	0.2	0	0
NON-TRIVIAL	4	1	0.3	0	0	1	0.3	0	0	1	0.3	0	0	1	0.3	0	0
Tireworld	8	1.2K	1.2K	1.2K	1.2K	1.2K	974.7	974.7	974.7	0.5	0.2	0.2	0.2	2.4	0.5	0.5	0.5
NON-TRIVIAL	7	1.7K	1.7K	1.7K	1.7K	1.7K	1.4K	1.4K	1.4K	0.4	0.2	0.2	0.2	2.7	0.5	0.5	0.5
Zenotravel	1	309.3	309.3	309.3	309.3	309.3	309.3	309.3	309.3	2.7	2.7	2.7	2.7	2.7	2.7	2.7	2.7
Probabilistic Resource-Constrained Benchmarks																	
NoMystery	4	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	433	430.8	433	430.8	432.6	418.8	432.6	418.8
Rovers	5	2.8K	2.8K	2.8K	2.8K	2.8K	2.8K	2.8K	2.8K	15.2	14.8	15.1	14.8	15.2	14.9	15.1	14.9
TPP	6	1.3K	1.3K	1.3K	1.3K	1.3K	1.3K	1.3K	1.3K	112.6	89.2	95.7	89.2	112.6	89.2	95.7	89.2
NON-TRIVIAL	5	2.3K	2.3K	2.3K	2.3K	2.3K	2.3K	2.3K	2.3K	149.2	127.2	138.5	127.2	149.2	127.2	138.5	127.2

Table 8: Cyclic planning. Top: MaxProb geometric mean runtime (in CPU seconds). Bottom: MaxProb geometric mean search space size (number of states visited) in multiples of 1000. Similar setup and presentation as in Table 4: "#" gives the size of the instance basis. The default are commonly solved instances, skipping trivial ones. "NON-TRIVIAL" uses only those instances not solved by VI in < 1 second. ("ONLY-H" not shown, see text.)

the search space sizes are exactly the same, and runtimes differ only by a few seconds. In difference to Tables 4 and 5, we do not include "ONLY-H" rows, because this would not be interesting here: FRET-V^U hardly solves more instances than VI, so would have to be excluded from these rows; but then, the data would compare only LRTDP vs. HDP, which perform very similarly anyway.

Most striking in Table 8 is the consistency with which, and the extent by which, FRET-π^U visits less states than its competitors (for both LRTDP and HDP). This advantage typically yields better runtimes as well, with the notable exception of NoMystery, where the larger number of FRET iterations results in a substantial slow-down, despite the much smaller search space: While FRET-V^U with LRTDP only requires 11 FRET iterations on average, in the NoMystery instances commonly solved with FRET-π^U and LRTDP, the latter configuration requires over 20000 iterations on average. Similarly when using HDP.

The impact of dead-end pruning is notably smaller than in the acyclic case: search spaces are reduced substantially in only a single domain, ExplodingBlocks. In the other domains, there either is no reduction, or a minor/moderate one only.

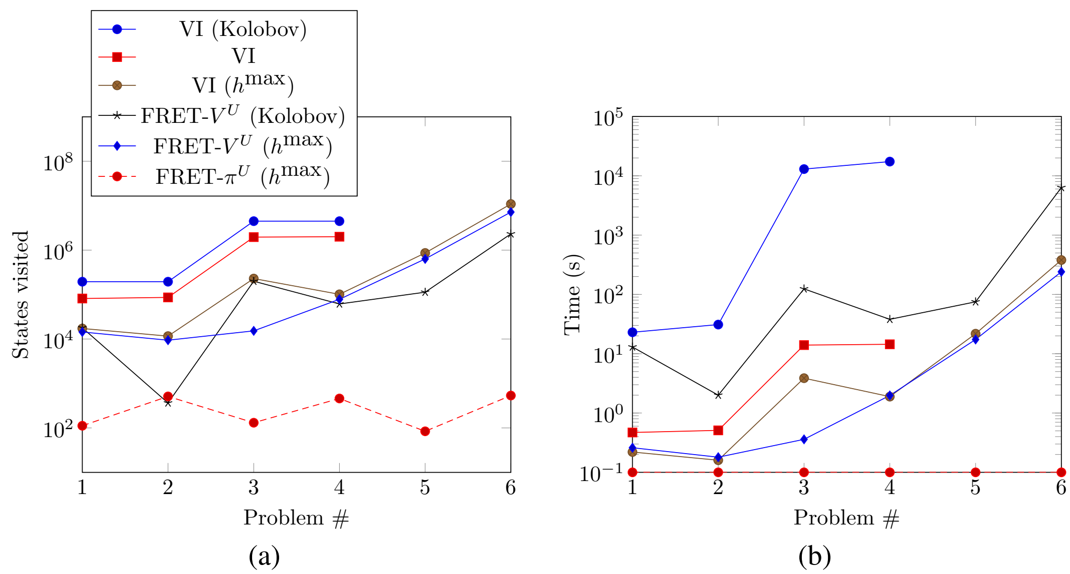


Figure 9: Cyclic planning. Results on ExplodingBlocks, as shown by Kolobov et al. (2011): FRET vs VI, (a) number of states visited, (b) runtime in CPU seconds, as a function of the IPPC instance index. Different variants included for comparison. The data for Kolobov et al. is taken from their paper (as this code is not available anymore), hence the runtime comparison is modulo the different computational platforms, and should be treated with care. All shown FRET configurations use LRTDP$|_{\mathrm{U}}$, with default node selection.

ExplodingBlocks also happens to be the single domain Kolobov et al. (2011) experimented with. Figure 9 provides a detailed comparison to Kolobov et al.'s data, which is the only state of the art measure provided by previous work. We use here the exact runtime/search space size data reported by Kolobov et al.; recall that their source code is not available anymore.

Kolobov et al. (2011) ran VI with no pruning vs. FRET-V^U using LRTDP with pruning based on SixthSense (Kolobov et al., 2010). They observed a coverage of 4 for the former and of 6 for the latter, identical with our results for VI "–" vs. FRET-V^U using LRTDP with $h^{\max}$. To give more detail, Figure 9 shows the number of states visited, and the total runtime, in terms of plots over IPPC instance index as done by Kolobov et al (2011).

Consider first Figure 9 (a), the search space size. The only difference between VI (Kolobov) and our VI here is the different task/state representation resulting from the respective implementation framework, the FD framework being somewhat more effective. The substantially better performance of VI with $h^{\max}$ dead-end pruning shows that the omission of Kolobov et al.'s (2011) study, using dead-end pruning in FRET but not in VI, indeed obfuscates the possible conclusions regarding the effect of heuristic search vs. the effect of the state pruning itself: *with* $h^{\max}$ *pruning, VI is almost as effective as FRET-*V^U *using the same pruning*. Kolobov et al.'s FRET-V^U also is very close to this, except for exploring significantly less states in the large instances. The latter shows, especially given the more effective representation in FD, that SixthSense is a stronger dead-end detector here than $h^{\max}$. That is hardly surprising, considering the information sources in SixthSense – outcomes of (determinized) classical planning for guidance, and h^2 (Graphplan) based validity tests.

On the other hand, SixthSense's information sources are much more time-intensive than $h^{\max}$, which presumably is the reason for the runtime picture in Figure 9 (b). The latter is qualitatively very similar to (a), except that FRET-V^U (Kolobov) is significantly *worse*, rather than better, on the largest instance. This last conclusion should be taken with a grain of salt though, given the different computational environments.

Certainly, given the clarity of FRET-π^U's advantage in both search space size and runtime, one can conclude that this variant of FRET substantially improves over the previous state of the art.

7.3.2 (2) ATLEASTPROB AND APPROXPROB PARAMETER ANALYSIS

For the weaker objectives AtLeastProb and ApproxProb, as before we examine coverage as a function of θ respectively δ. Figure 10 shows the data.

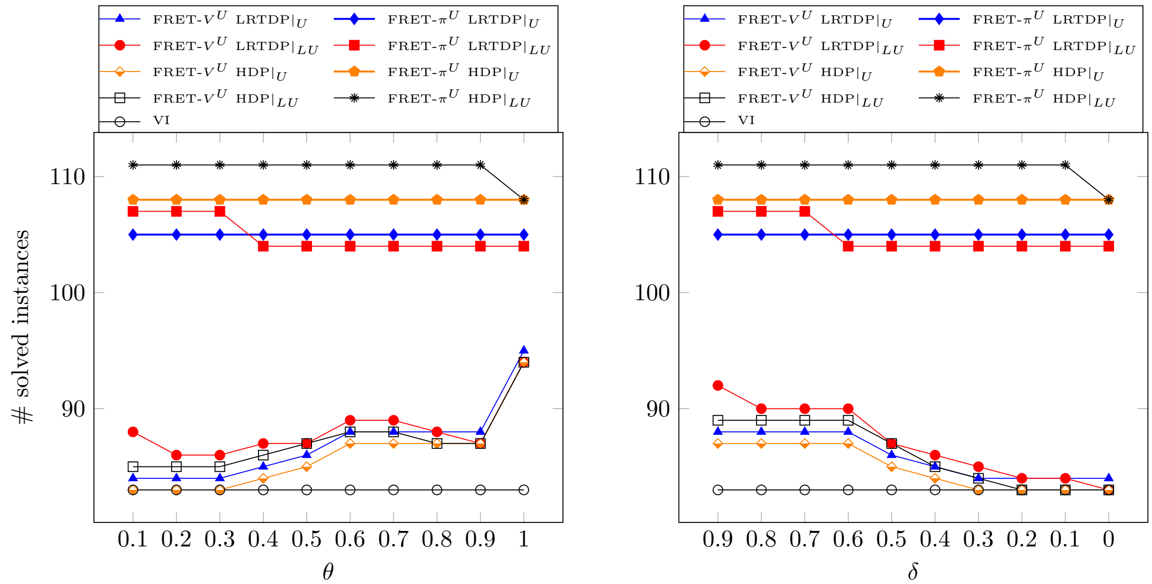
For FRET-V^U, the behavior in Figure 10 is similar to that for the acyclic case in Figure 5. In particular, when maintaining both an upper and a lower bound, FRET-V^U exhibits an easy-hard-easy pattern due to the advantages of early termination.

For FRET-π^U, though, the curves are flat over θ, and the only observation is a small advantage of using V^L in addition to V^U. This is due to the scaling of benchmarks, combined with an extreme performance loss at some point in the scaling: in each domain, there is an instance number x so that, below x, FRET-π^U can solve all instances completely (i.e., solving MaxProb), while above x neither $V^L(I)$ nor $V^U(I)$ can be improved at all, remaining 0 respectively 1 up to the time/memory limit. On smaller instances, we do get the expected anytime behavior. Figure 11 exemplifies this. The easy-hard-easy pattern would thus emerge for smaller runtime/memory limits.[14]

8. Conclusion

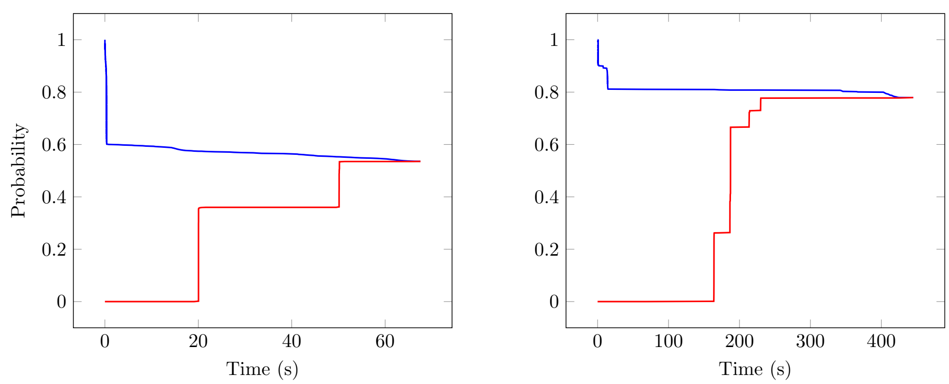
Optimal goal probability analysis in probabilistic planning is a notoriously hard problem, to the extent that the amount of work addressing it is limited. Our investigation contributes a comprehensive design space of known and adapted algorithms addressing this problem, designing several new algorithm variants along the way, and establishing an FD implementation basis supporting the tight integration of MDP heuristic search with classical planning techniques. Our experiments clarify the

14. Figure 11 (b) considers the largest instance feasible when using $h^{\max}$ pruning. Figure 11 (a) considers the second-largest instance feasible without pruning: on the largest one feasible without pruning, namely instance 05, the maximum goal probability is 1 so the anytime curve for V^U is not interesting.



(a) (b)

Figure 10: Cyclic planning. Total coverage for AtLeastProb as a function of θ in (a), for ApproxProb as a function of δ in (b). All configurations use default node selection and $h^{\max}$ dead-end pruning.



(a) (b)

Figure 11: Cyclic planning. Anytime behavior of FRET-π^U with LRTDP$|_{\mathrm{LU}}$and HDP$|_{\mathrm{LU}}$, with default node selection, (a) without pruning for ExplodingBlocks instance 04, and (b) with $h^{\max}$ pruning for instance 15.

empirical state of the art, and exhibit substantial improvements thanks to new techniques and technique combinations. They furthermore showcase the opportunities arising from naturally acyclic problems, and from early termination on criteria weaker than maximum goal probability.

We hope that these encouraging results and new implementation basis will inspire renewed interest and research in this important problem. There are many promising future directions, of which we would like to emphasize:

- *Advanced admissible goal probability estimators.* These could be obtained, e.g. from abstractions interpreted as bounded-parameter MDPs (Givan, Leach, & Dean, 2000). A promising approach is to extend state-of-the-art classical-planning abstraction techniques – pattern databases (Edelkamp, 2001; Haslum, Botea, Helmert, Bonet, & Koenig, 2007), merge-and-shrink (Helmert et al., 2014), Cartesian abstractions (Seipp & Helmert, 2013, 2014) – to the probabilistic setting.

- *Hybrids of heuristic search with Monte-Carlo tree search.* This appears a promising option to improve anytime behavior, with respect to the upper and/or lower bound, and thus foster early termination. Inspiration could be taken here from existing such hybrids, geared toward other purposes (Keller & Eyerich, 2012; Bonet & Geffner, 2012; Keller & Helmert, 2013).

- *Exploiting dominance relations.* Goal probability can only be higher in dominating states, raising the opportunity to prune dominated regions and/or transfer upper/lower bounds across states. State domination is ubiquitous in limited-budget planning (and resource-constrained planning). More general domination relations have been shown to exist also in many other classical planning problems (Torralba & Hoffmann, 2015), and the transfer of these techniques to the probabilistic case, via all-outcomes determinization, should be straightforward.

Last but not least, simulated penetration testing is an application worth algorithms research in its own right. The basic idea is to exploit the particular structure of such models, specifically their partially delete-relaxed behavior. A characterizing property of simulated penetration testing is that any action, once applicable, remains applicable until it is first executed (once the attacker gets into a position enabling an exploit, that exploit remains enabled). Hence, like in delete-relaxed planning, to find an optimal solution, naïvely we will branch over that same action at every state ever after. To combat this, there are at least three interesting directions. Following Pommerening and Helmert's (2012) methods for computing h^+, different branching schemes might apply, the challenge being to maintain value function correctness. Following Gefen and Brafman's (2012) methods for computing h^+, partial-order reduction could be adapted, the challenge being to deal with the action interference entailed by a shared budget. Finally, methods specific to the probabilistic setting may apply: intuitively, to preserve optimality, certain actions need to be attempted only if an alternate goal path failed. This suggests to identify, and branch at, only particular "critical points" along any search path.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation", as well as by the Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) under grant 16KIS0656. We thank Christian Muise for his Probabilistic-PDDL extension of the FD parser. We thank Andrey Kolobov for discussions. We thank the anonymous reviewers, whose comments helped to improve the paper.

Appendix A. Depth-First Heuristic Search for Cyclic Problems

The pseudo-code of the family of depth-first heuristic search algorithms (DFHS) for general (cyclic) probabilistic planning problems is shown in Figure 12.

```
procedure GoalProb-DFHS
Θ := {I};
loop do
  [early termination criteria exactly as in GoalProb-AO*]
  if (Label and I is not labeled as solved)
    or (VI and π^U changed after running VI on the π^U-greedy graph) then
    index := 0
    DFHS-Exploration(I)
    set IDX of visited states to ∞
    clean stack and visited
  else return π^U endif /* regular termination */
endloop
procedure DFHS-Exploration(s):
  if s ∉ Θ then Initialize(s) endif
  if s ∈ S_⊤ ∪ S_⊥ or s is labeled solved then
    label s solved
    return ⊥
  endif
  flag := ⊥
  if FW then
    if V^U(s) is not ε consistent then flag := ⊤ endif
    update V^U(s), π^U(s), V^L(s), π^L(s)
    if Consist and flag then return ⊤ endif
  endif
  s.IDX := index; s.lowlink := index
  push s onto stack; mark s as visited
  index := index + 1
  foreach t with P(s, π^U(s), t) > 0 do
    if t.IDX = ∞ then
      flag := DFHS-Exploration(s) ∨ flag
      if t.IDX < ∞ and t.lowlink < s.lowlink then s.lowlink := t.lowlink endif
    else if t is on stack and t.IDX < s.lowlink then s.lowlink := t.IDX endif
  done
  if flag or ¬FW then
    if V^U(s) is not ε-consistent then flag := ⊤ endif
    update V^U(s), π^U(s), V^L(s), and π^L(s)
  endif
  if Label and ¬flag and s.IDX = s.lowlink then
    while forever
      t := stack.pop()
      label t solved
      if t = s then break endif
    done
  endif
  return flag
```

Figure 12: Depth-First Heuristic Search (DFHS) for general (cyclic) MaxProb, AtLeastProb, and ApproxProb.

Appendix B. Landmarks Pruning: Admissible Heuristic vs. Budget Reduction

As stated, Domshlak and Mirkis' (2015) problem reformulation, pruning states based on a global budget reduced using disjunctive action landmarks, is equivalent, regarding the states pruned by the method on its own, to the much simpler method using the same landmarks for pruning against the remaining original budget. We now give this argument, previously made only for unit costs and pairwise disjoint landmarks, for the general setting. We assume a classical planning setup for simplicity. The arguments in probabilistic and oversubscription setups are essentially the same.

Assume a STRIPS planning task $\Pi = (F, A, I, G)$, with action costs $c(a)$ and with a global budget b. We use a notation following admissible landmark heuristics as per Karpas and Domshlak (2009). Let L be a set of disjunctive action landmarks for I, i.e., for every $l \in L$ and every action sequence $\vec{a}$ leading from I to the goal, $\vec{a}$ touches l (there exists $a \in l$ used on $\vec{a}$). Let furthermore $cp : A \times L \mapsto \mathbb{R}_0^+$ be a cost partitioning, i.e., a function satisfying, for each $a \in A$, that $\sum_{l \in L} c(a, l) \leq c(a)$. Denote $h(l) := \min_{a \in l} cp(a, l)$, and for a subset $L' \subseteq L$ of landmarks denote $h(L') := \sum_{l \in L'} h(l)$. Intuitively, each landmark $l \in L$ is assigned a weight $h(l)$ via cp, and the admissible heuristic value $h(L)$ for I is obtained by summing up these weights.

We now describe Domshlak and Mirkis' (2015) pruning technique in these terms. Domshlak and Mirkis' formulation is in terms of a compilation into a planning language, which is more complicated, but is equivalent to our formulation here as far as the pruning is concerned.

Domshlak and Mirkis' technique maintains the "non-used" landmarks as part of states. Namely, for a state s reached on path $\vec{a}$, $l \in L$ is non-used in s iff $\vec{a}$ does not touch l. We denote the set of non-used landmarks in s by $L(s)$. Obviously, the $l \in L(s)$ are landmarks for s. Note also that, as $L(s)$ is part of the state, even if two search paths lead to the same end state but use different landmarks, their end states are considered to be different. This restriction arises from the compilation approach, where the book-keeping of landmarks must happen inside the language, i.e., inside states. One could formulate the pruning technique without this restriction; we get back to this below.

The pruning technique now arises from the interplay of a reduced global budget and reduced action costs depending on non-used landmarks. Define the reduced global budget as $b' := b - h(L)$. For any action a, denote by $L(a)$ the set of landmarks a participates in, i.e., $L(a) := \{l \mid l \in L, a \in l\}$. For any state t during search, and an applicable action a, the transition from t to $t[[a]]$ has a reduced cost, namely the cost $c(a) - h(L(a) \cap L(t))$. In words, we reduce the cost of a by the (summed-up) weight of the non-used landmarks a participates in.

Consider now some state s during search. Denote the remaining reduced budget in s by $b'(s)$. Say that we prune s iff $b'(s) < 0$.[15] Consider any path $\vec{a}$ ending in s. As non-used landmarks are part of the state, all these paths must touch the same subset of landmarks from L, namely $L \setminus L(s)$. Denote the actual cost of $\vec{a}$ by $c(\vec{a}) := \sum_{a \in \vec{a}} c(a)$. Relative to this cost, the cost saved thanks to the cost reduction is exactly $h(L \setminus L(s))$, the weight of the touched landmarks. Therefore, $b'(s) = b' - (c(\vec{a}) - h(L \setminus L(s))) = (b - h(L)) - c(\vec{a}) + h(L \setminus L(s))$. By definition of h, this equals $(b - \sum_{l \in L} h(l)) - c(\vec{a}) + \sum_{l \in L \setminus L(s)} h(l)$, which equals $b - c(\vec{a}) - \sum_{l \in L(s)} h(l) = b - c(\vec{a}) - h(L(s))$. Thus, s is pruned, $b'(s) < 0$, iff $b - c(\vec{a}) < h(L(s))$. The latter condition is the same as $b(s) < h(L(s))$, which is exactly the pruning condition resulting from using $h(L(s))$ as an admissible heuristic function pruning against the remaining budget.

In a non-compilation setting, one could, as is indeed customary in admissible landmark heuristics, handle landmarks in a path-dependent manner. That is, non-used landmarks are maintained as

15. Domshlak and Mirkis (2015) do not maintain the remaining budget as part of the state, but instead prune if $g(s) > b'$. This is, obviously, equivalent, except that duplicate detection is more powerful as it compares states based on their facts $F(s)$ only. For the purpose of our discussion here, this does not make a difference. Note that, in the probabilistic setting, we do have to distinguish states based on both $F(s)$ and $b(s)$, as goal probability depends on both so maintaining only the best way of reaching $F(s)$ does not suffice to compute the exact goal probability of the initial state.

annotations to states rather than as part of them, and multiple search paths may end in the same state s but use different landmarks. The set of remaining landmarks $L(s)$ for s then is the union over those for each individual path; that is, $l \in L$ is non-used in s iff there exists at least one path that does not touch l. This still suffices to show that l is a landmark for s. The landmark heuristic approach as per Karpas and Domshlak (2009) does this kind of book-keeping, and uses the admissible heuristic value $h(L(s))$.

If one were to apply Domshlak and Mirkis' (2015) reformulation technique without maintaining landmarks as part of state, then the notion of transition-cost reduction would have to become more complicated (lest one loses information). This is because, if s is reached on $\vec{a_1}$ with a reduced cost due to touching landmark l_1, but later on we find another path $\vec{a_2}$ to s that does not touch l_1, then l_1 actually still is a valid landmark for s, and therefore there was no need to reduce the cost on $\vec{a_1}$. To account for this, we would have to revise path costs posthoc, every time a new path to s becomes available. After these revisions, the cost reduction on each path $\vec{a}$ to s is exactly $h(L \setminus L(s))$: the weight of the non-used landmarks $L(s)$ is no longer subtracted, and the weight of the other landmarks $L \setminus L(s)$ is subtracted on every $\vec{a}$ because, by definition, every $\vec{a}$ touches every $l \in L \setminus L(s)$. So the cost saved on every path $\vec{a}$ to s, relative to $\vec{a}$, is exactly $h(L \setminus L(s))$, from which point the same arguments as above apply to show that the pruning is equivalent to pruning via $b(s) < h(L(s))$. (This is a stronger pruning method than what we would get without posthoc path cost revision.)

In summary, s based on reduced remaining budget $b'(s) < 0$ is equivalent to pruning s based on original remaining budget vs. the landmark heuristic $b(s) < h(L(s))$. It should be noted, though, that such pruning is not the only benefit of Domshlak and Mirkis' (2015) reformulation technique. The technique allows to compute another, complementary, admissible heuristic h on the reformulated task Π' (and this is what Domshlak and Mirkis point out as part of the motivation, and what they do in practice). From our perspective here, the landmark heuristic and h are used *additively* for admissible pruning against the remaining budget, where additivity is achieved with a method generalizing cost partitionings: in Π', the cost-reduced variant of each action can be applied only once. So if h does not abstract away this constraint, and if h uses an action twice, then it employs the reduced cost only once, yet pays the full cost the second time. Exploring this kind of generalized cost partitioning in more detail is an interesting research line for future work.

References

Altman, E. (1999). *Constrained Markov Decision Processes*. CRC Press.

Baier, C., Größer, M., Leucker, M., Bollig, B., & Ciesinski, F. (2004). *Controller Synthesis for Probabilistic Systems (Extended Abstract)*, pp. 493–506. Springer US, Boston, MA.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control, (2 Volumes)*. Athena Scientific.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neurodynamic Programming*. Athena Scientific.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1–2), 5–33.

Bonet, B., & Geffner, H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. In Gottlob, G. (Ed.), *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 1233–1238, Acapulco, Mexico. Morgan Kaufmann.

Bonet, B., & Geffner, H. (2003b). Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E., Muscettola, N., & Nau, D. (Eds.), *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pp. 12–21, Trento, Italy. Morgan Kaufmann.

Bonet, B., & Geffner, H. (2005). mgpt: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, *24*, 933–944.

Bonet, B., & Geffner, H. (2006). Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs. In Long, D., & Smith, S. (Eds.), *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pp. 142–151, Ambleside, UK. Morgan Kaufmann.

Bonet, B., & Geffner, H. (2012). Action selection for MDPs: Anytime AO* versus UCT. In Hoffmann, J., & Selman, B. (Eds.), *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, Toronto, ON, Canada. AAAI Press.

Bryce, D., & Buffet, O. (2008). 6th international planning competition: Uncertainty part. In *Proceedings of the 6th International Planning Competition (IPC'08)*.

Camacho, A., Muise, C., & McIlraith, S. A. (2016). From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., & Sanner, S. (Eds.), *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*. AAAI Press.

Chatterjee, K., Chmelik, M., Gupta, R., & Kanodia, A. (2015). Optimal cost almost-sure reachability in POMDPs. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pp. 3496–3502. AAAI Press.

Chatterjee, K., Chmelik, M., Gupta, R., & Kanodia, A. (2016). Optimal cost almost-sure reachability in POMDPs. *Artificial Intelligence*, *234*, 26–48.

Coles, A. J. (2012). Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In Raedt, L. D. (Ed.), *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pp. 252–257, Montpellier, France. IOS Press.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2013). A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research*, *46*, 343–412.

Coles, A. J., Coles, A., García Olaya, A., Jiménez, S., Linares López, C., Sanner, S., & Yoon, S. (2012). A survey of the seventh international planning competition. *The AI Magazine*, *33*(1).

Dai, P., Mausam, Weld, D. S., & Goldsmith, J. (2011). Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, *42*, 181–209.

Dean, T. L., & Givan, R. (1997). Model minimization in markov decision processes. In Kuipers, B. J., & Webber, B. (Eds.), *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI'97)*, pp. 106–111, Portland, OR. MIT Press.

Domshlak, C., & Mirkis, V. (2015). Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *Journal of Artificial Intelligence Research*, *52*, 97–169.

Dräger, K., Finkbeiner, B., & Podelski, A. (2009). Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer*, *11*(1), 27–37.

Edelkamp, S. (2001). Planning with pattern databases. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the 6th European Conference on Planning (ECP'01)*, pp. 13–24. Springer-Verlag.

Gefen, A., & Brafman, R. I. (2012). Pruning methods for optimal delete-free planning. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Givan, R., Leach, S. M., & Dean, T. (2000). Bounded-parameter Markov decision processes. *Artificial Intelligence*, *122*(1-2), 71–109.

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

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In Howe, A., & Holte, R. C. (Eds.), *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pp. 1007–1012, Vancouver, BC, Canada. AAAI Press.

Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the 6th European Conference on Planning (ECP'01)*, pp. 121–132. Springer-Verlag.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 162–169. AAAI Press.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, *61*(3).

Hoffmann, J. (2015). Simulated penetration testing: From "Dijkstra" to "Turing Test++". In Brafman, R., Domshlak, C., Haslum, P., & Zilberstein, S. (Eds.), *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

Hoffmann, J., Kissmann, P., & Torralba, Á. (2014). "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T. (Ed.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, Prague, Czech Republic. IOS Press.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Hou, P., Yeoh, W., & Varakantham, P. (2014). Revisiting risk-sensitive MDPs: New algorithms and results. In Chien, S., Do, M., Fern, A., & Ruml, W. (Eds.), *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Jimenez, S., Coles, A., & Smith, A. (2006). Planning in probabilistic domains using a deterministic numeric planner. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig'06)*.

Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. In Boutilier, C. (Ed.), *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 1728–1733, Pasadena, California, USA. Morgan Kaufmann.

Katz, M., Hoffmann, J., & Helmert, M. (2012). How to relax a bisimulation?. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 101–109. AAAI Press.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Keller, T., & Helmert, M. (2013). Trial-based heuristic tree search for finite horizon MDPs. In Borrajo, D., Fratini, S., Kambhampati, S., & Oddi, A. (Eds.), *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, Rome, Italy. AAAI Press.

Kolobov, A. (2013). *Scalable Methods and Expressive Models for Planning Under Uncertainty*. Ph.D. thesis, University of Washington.

Kolobov, A., Mausam, & Weld, D. S. (2010). Sixthsense: Fast and reliable recognition of dead ends in MDPs. In Fox, M., & Poole, D. (Eds.), *Proceedings of the 24th National Conference of the American Association for Artificial Intelligence (AAAI'10)*, Atlanta, GA, USA. AAAI Press.

Kolobov, A., Mausam, & Weld, D. S. (2012). A theory of goal-oriented MDPs with dead ends. In de Freitas, N., & Murphy, K. P. (Eds.), *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI'12)*, pp. 438–447, Catalina Island, CA, USA. AUAI Press.

Kolobov, A., Mausam, Weld, D. S., & Geffner, H. (2011). Heuristic search for generalized stochastic shortest path MDPs. In Bacchus, F., Domshlak, C., Edelkamp, S., & Helmert, M. (Eds.), *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*. AAAI Press.

Kurniawati, H., Hsu, D., & Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems IV*.

Kuter, U., & Hu, J. (2007). Computing and using lower and upper bounds for action elimination in MDP planning. In Miguel, I., & Ruml, W. (Eds.), *Proceedings of the 7th International Symposium on Abstraction, Reformulation, and Approximation (SARA-07)*, Vol. 4612 of *Lecture Notes in Computer Science*, Whistler, Canada. Springer-Verlag.

Kwiatkowska, M., Parker, D., & Qu, H. (2011a). Incremental quantitative verification for markov decision processes. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pp. 359–370.

Kwiatkowska, M. Z., Norman, G., & Parker, D. (2011b). Prism 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., & Qadeer, S. (Eds.), *Proceedings of the 23rd International on Conference Computer Aided Verification (CAV'11)*, Vol. 6806 of *Lecture Notes in Computer Science*, pp. 585–591. Springer.

Little, I., Aberdeen, D., & Thiébaux, S. (2005). Prottle: A probabilistic temporal planner. In Veloso, M. M., & Kambhampati, S. (Eds.), *Proceedings of the 20th National Conference of the American Association for Artificial Intelligence (AAAI'05)*, pp. 1181–1186, Pittsburgh, Pennsylvania, USA. AAAI Press.

Little, I., & Thiebaux, S. (2007). Probabilistic planning vs replanning. In *ICAPS Workshop on the International Planning Competition: Past, Present and Future*.

Marecki, J., & Tambe, M. (2008). Towards faster planning with continuous resources in stochastic domains. In Fox, D., & Gomes, C. (Eds.), *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, pp. 1049–1055, Chicago, Illinois, USA. AAAI Press.

McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)*.

Meuleau, N., Benazera, E., Brafman, R. I., Hansen, E. A., & Mausam, M. (2009). A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, *34*(1), 27–59.

Milner, R. (1990). Operational and algebraic semantics of concurrent processes. In van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pp. 1201–1242. Elsevier and MIT Press.

Muise, C. J., McIlraith, S. A., & Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Nakhost, H., Hoffmann, J., & Müller, M. (2012). Resource-constrained planning: A monte carlo random walk approach. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 181–189. AAAI Press.

Nilsson, N. J. (1971). *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill.

Nissim, R., Hoffmann, J., & Helmert, M. (2011). Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 1983–1990. AAAI Press/IJCAI.

Pommerening, F., & Helmert, M. (2012). Optimal planning for delete-free tasks with incremental lm-cut. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Richter, S., & Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 273–280. AAAI Press.

Sanner, S. (2010). Relational dynamic influence diagram language (rddl): Language description. Available at http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Santana, P., Thibaux, S., & Williams, B. (2016). RAO*: An algorithm for chance-constrained POMDPs. In Schuurmans, D., & Wellman, M. (Eds.), *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*, pp. 3308–3314. AAAI Press.

Sarraute, C., Buffet, O., & Hoffmann, J. (2012). POMDPs make better hackers: Accounting for uncertainty in penetration testing. In Hoffmann, J., & Selman, B. (Eds.), *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pp. 1816–1824, Toronto, ON, Canada. AAAI Press.

Seipp, J., & Helmert, M. (2013). Counterexample-guided Cartesian abstraction refinement. In Borrajo, D., Fratini, S., Kambhampati, S., & Oddi, A. (Eds.), *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pp. 347–351, Rome, Italy. AAAI Press.

Seipp, J., & Helmert, M. (2014). Diverse and additive cartesian abstraction heuristics. In Chien, S., Do, M., Fern, A., & Ruml, W. (Eds.), *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Smith, T., & Simmons, R. G. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In Gil, Y., & Mooney, R. J. (Eds.), *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI'06)*, pp. 1227–1232, Boston, Massachusetts, USA. AAAI Press.

Steinmetz, M., Hoffmann, J., & Buffet, O. (2016). Revisiting goal probability analysis in probabilistic planning. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., & Sanner, S. (Eds.), *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*. AAAI Press.

Tarjan, R. E. (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing*, *1*(2), 146–160.

Teichteil-Königsbuch, F. (2012). Stochastic safest and shortest path problems. In Hoffmann, J., & Selman, B. (Eds.), *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, Toronto, ON, Canada. AAAI Press.

Teichteil-Königsbuch, F., Kuter, U., & Infantes, G. (2010). Incremental plan aggregation for generating policies in MDPs. In van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., & Sen, S. (Eds.), *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pp. 1231–1238. IFAAMAS.

Teichteil-Königsbuch, F., Vidal, V., & Infantes, G. (2011). Extending classical planning heuristics to probabilistic planning with dead-ends. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, San Francisco, CA, USA. AAAI Press.

Torralba, Á., & Hoffmann, J. (2015). Simulation-based admissible dominance pruning. In Yang, Q. (Ed.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pp. 1689–1695. AAAI Press/IJCAI.

Yoon, S. W., Fern, A., & Givan, R. (2007). FF-Replan: a baseline for probabilistic planning. In Boddy, M., Fox, M., & Thiebaux, S. (Eds.), *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pp. 352–359, Providence, Rhode Island, USA. Morgan Kaufmann.

Younes, H. L. S., Littman, M. L., Weissman, D., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, *24*, 851–887.