Revisiting Goal Probability Analysis in Probabilistic Planning Technical Report

Marcel Steinmetz and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
{steinmetz,hoffmann}@cs.uni-saarland.de

Olivier Buffet
INRIA / Université de Lorraine
Nancy, France
olivier.buffet@loria.fr

Abstract

Maximizing goal probability is an important objective in probabilistic planning, yet algorithms for its optimal solution are severely underexplored. There is scant evidence of what the empirical state of the art actually *is*. Focusing on heuristic search, we close this gap with a comprehensive empirical analysis of known and adapted algorithms. We explore both, the general case where there may be 0-reward cycles, and the practically relevant special case of acyclic planning, like planning with a limited action-cost budget. We consider three different algorithmic objectives. We design suitable termination criteria, search algorithm variants, dead-end pruning methods using classical planning heuristics, and node selection strategies. Our evaluation on more than 1000 benchmark instances from the IPPC, resource-constrained planning, and simulated penetration testing reveals the behavior of heuristic search, and exhibits several improvements to the state of the art.

1 Introduction

Goal probability maximization in MDPs is important in planning scenarios ranging from critical decision-making (e. g. maximizing the probability to survive) to security tests (analyzing the chances that an attacker may compromise a valuable asset), and generally in problems with *unavoidable dead-ends* (e. g. (Kolobov et al. 2011; Kolobov, Mausam, and Weld 2012; Teichteil-Königsbuch 2012)). The objective partly underlies the International Probabilistic Planning Competition (IPPC) (Younes et al. 2005; Bryce and Buffet 2008; Coles et al. 2012), when planners are evaluated by how often they reach the goal in online policy execution.

We consider here the optimal offline setting, i. e., computing the exact maximum goal probability. We refer to this objective as *MaxProb*. While MaxProb certainly is relevant, there has been little work towards developing solvers. Optimal MDP heuristic search (Barto, Bradtke, and Singh 1995; Hansen and Zilberstein 2001; Bonet and Geffner 2003; McMahan, Likhachev, and Gordon 2005; Smith and Simmons 2006; Bonet and Geffner 2006) has been successful in expected-cost minimization, but suffers from a lack of admissible heuristic estimators of goal probability. The best known possibility is to detect *dead-ends* and set their estimate to 0, using the trivial estimate 1 elsewhere. Another major obstacle are complications arising from 0-reward cycles. As pointed out by Kolobov et al. (2011), MaxProb is equivalent to a non-discounted reward maximization problem, not fitting the stochastic shortest path (SSP) framework (Bertsekas 1995) because non-goal cycles receive 0 reward and thus improper policies do not accumulate reward $-\infty$.

Kolobov et al. propose *FRET (find, revise, eliminate traps)*, which admits heuristic search, but necessitates several iterations of complete searches, in between which FRET eliminates 0-reward cycles (traps). Hou et al. (2014) consider several variants of topological VI (Dai et al. 2011), solving MaxProb but necessitating to build the entire reachable state space. Kolobov et al. (2012) and Teichteil (2012) consider objectives asking for the cheapest policy among those maximizing goal probability, also requiring FRET or VI. Other works addressing goal probability maximization (e. g. (Teichteil-Königsbuch, Kuter, and Infantes 2010; Camacho, Muise, and McIlraith 2016)) do not aim at guaranteeing optimality. In summary, heuristic search for MaxProb is challenging, and has only been addressed by Kolobov et al. (2011).

Kolobov et al.'s experiments run only one configuration of search (LRTDP (Bonet and Geffner 2003)), with one possibility for dead-end detection (SixthSense (Kolobov, Mausam, and Weld 2010)), on a single domain (Exploding-Blocks). This outperforms VI, but the dead-end detection is not used in VI so it is unclear to what extent this is due to the actual heuristic search, rather than the state pruning itself.

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

Question (ii) is interesting because such special cases and weaker objectives do indeed exist. A practically relevant special case is probabilistic planning with acyclic state spaces. This applies, e. g., to IPPC TriangleTireworld. More importantly, planning with a limited action-cost budget, *limited-budget planning*, is acyclic when action costs are non-0, strictly decreasing the remaining budget. Furthermore, simulated *penetration testing (pentesting)*, as per Hoffmann (2015), is acyclic. The MDP there models a network intrusion from the point of view of an attacker, which is acyclic because each exploit can be attempted at most once.

In acyclic problems, there are no 0-reward cycles so we are facing an SSP problem and the need for FRET disappears.

Regarding weaker objectives, in addition to MaxProb, it is relevant 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. For example, in pentesting AtLeastProb naturally assesses the level of network security: Can an attacker reach a target host (e. g. a customer data server) with probability greater than a given security margin (e. g. 0.01)?

Both AtLeastProb and ApproxProb allow early termination based on maintaining a lower (pessimistic) bound V^L in addition to the 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.

To answer our research questions (i) and (ii), we design an algorithm space characterized by (a) *search algorithm*, (b) *dead-end pruning method*, and (c) *node selection strategy*. For (a), we design variants of AO* (Nilsson 1971)and LRTDP (Bonet and Geffner 2003) maintaining upper and lower bounds for early termination, and we design a variant of FRET better suited to problems with uninformative initial upper bounds. We furthermore design a new probabilistic-state-space reduction method, via determinized bisimulation. For (b), we employ classical-planning heuristic functions, a connection not made before and which is especially promising in limited-budget planning where we can prune against the remaining budget.[1] For (c), we design a comprehensive arsenal of simple strategies biasing node selection to foster early termination.

Our techniques are implemented in FD (Helmert 2006). We explore their behavior on a benchmark suite including domains from the IPPC, resource-constrained planning, and pentesting, comprising 1089 instances in total. Amongst other things, we observe: substantial benefits of heuristic search, even with trivial initial estimates (+9% total coverage), more so with initial estimates based on dead-end detection (+12%); substantial benefits of early termination (e. g. for AtleastProb +8% with $\theta = 0.2$ and +7% with $\theta = 0.9$); and dramatic benefits of our FRET variant (+32%). Our state-space reduction method yields an optimal MaxProb solver that scales just as well in TriangleTireworld as the sub-optimal solver Prob-PRP (Muise, McIlraith, and Beck 2012; Camacho, Muise, and McIlraith 2016) – yet not only for the standard version where the goal can be achieved with certainty, but also for the limited-budget version where that is not so.

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 variants of VI, AO*, LRTDP, and FRET. Section 4 describes our probabilistic-state-space reduction method, Section 5 describes the dead-end pruning methods, Section 6 describes the node selection strategies. Section 7 summarizes our experiments, and we conclude in Section 8. The main text omits some technical details, which we give Appendix A.

[1] On the side, we discover that Domshlak and Mirkis' (2015) landmarks compilation is, per se, equivalent to such pruning.

2 MDP Models

We consider a probabilistic extension of STRIPS, in two variants, with respectively 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 being 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 $\sum_{o \in O(a)} p(o) = 1$.

The *state space* of a task Π 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 $G \subseteq F(s)$. The *transition probability* function $P : S \times A \times S \mapsto [0, 1]$ is defined as follows. Action a is *applicable* to state s if $s \notin S_\top$ (goal states are absorbing) and $pre(a) \subseteq F(s)$. By $s[\![o]\!]$ we denote the result of outcome o in s, i. e., $F(s[\![o]\!]) = (F(s) \cup add(o)) \setminus del(o)$. $P(s, a, t)$ is $p(o)$ if a is applicable to s and $t = s[\![o]\!]$,[2] and is 0 otherwise (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 $S_\bot$.

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 with a *budget* $b \in \mathbb{R}_0^+$, and each outcome o being associated with a *cost* $c(o) \in \mathbb{R}_0^+$. In addition to their true facts $F(s)$, states s are 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 are lost, $s \in S_\bot$, because: the goal states $s \in S_\top$ are those where $G \subseteq F(s)$ *and* $b(s) \geq 0$; 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 budget, i. e., $b(s[\![o]\!]) = b(s) - c(o)$.

Note here that, if $c(o) > 0$ for all o, then the state space is acyclic because every transition strictly reduces the remaining budget. The state space is infinite due to the continuous state variable $b(s)$. The 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.

[2] 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).

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 and Mirkis 2015). A classical-planning variant would relate to resource-constrained planning (e. g. (Haslum and Geffner 2001; Nakhost, Hoffmann, and Müller 2012; Coles et al. 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 and Tambe 2008; Meuleau et al. 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, arguably limited-budget probabilistic planning is quite natural. Decision making is often constrained by a finite budget, and non-0 costs are often reasonable to assume. For example, any problem asking to achieve a goal within a given number of steps falls into this category.

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 $*$. The latter 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, and 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. A policy is *proper* if, from every state s on which π is defined, π eventually reaches an absorbing state with probability 1.

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:

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

The value of state s is $V^*(s) = \max_{\pi:\pi \text{ closed for } s} V^\pi(s)$. We don't need to exclude improper π from this maximization as, in difference to Kolobov et al.'s Generalized SSP problems, there are no negative rewards.

For acyclic state spaces, we are facing an SSP problem (every run ends in an absorbing state in a finite number of steps). For cyclic state spaces, the Bellman update operator may have multiple sub-optimal fixed points, and updates from an optimistic (upper-bound) initialization are not guaranteed to converge to the optimum. One can either use a pessimistic initialization, or Kolobov et al.'s FRET method.

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 next examine search algorithms, pruning methods, and node selection strategies, to solve these problems.

3 Search Algorithms

We use value iteration (VI) as a baseline, and we design variants of AO* and LRTDP. We furthermore design a variant of FRET better suited to problems with uninformative initial upper bounds. We consider the algorithms in this order.

3.1 VI

For VI, we make one forward pass building the reachable state space (actually its pruned subset, see Section 5). We initialize the value function 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. For the general case, we assume a parameter ϵ and run topological VI (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*, i. e. when its Bellman residual is at most ϵ.

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 to build 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 and Zilberstein 2001), disappears. Figure 1 shows pseudo-code. 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, Likhachev, and Gordon 2005; Little, Aberdeen, and Thiébaux 2005; Smith and Simmons 2006)), we maintain two value functions, namely both an upper bound V^U and a lower bound V^L on goal probability. Both are initialized trivially, for lack of heuristic estimators of goal probability (dead-end detection, as a simple but non-trivial V^U initialization, will be discussed in the next section). Nevertheless, both bounds can be useful for search. To refute an action, it often suffices to reduce V^U for just one of its 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

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

occurs frequently. Furthermore, there are various possibilities for 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 $V^U < \theta$. In ApproxProb, clearly we can terminate when $V^U(I) - V^L(I) \leq \delta$.[3]

Regarding correctness: Trivially, $V^U(s)$ and $V^L(s)$ indeed are upper respectively lower bounds on the goal probability of the states s in Θ, at any point in time. Furthermore, 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* on those states). Its goal probability $V^{\pi^L}(s)$ is at least the lower-bound goal probability, $V^{\pi^L}(s) \geq V^L(s)$, because $V^L(s)$ is monotonic.

[3]Observe that the $V^L = 1$ (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.

```
procedure $GoalProb\text{-}LRTDP$
$\Theta := \{I\}$; $Initialize(I)$
loop do
  [early termination criteria exactly as in GoalProb-AO$^*$]
  if $I$ is not labeled as solved then
     $LRTDP\text{-}Trial(I)$
  else return $\pi^U$ endif /* regular termination */
endloop
procedure $LRTDP\text{-}Trial(s)$:
  $P$ := empty stack
  while $s$ is not labeled as solved do
     push $s$ onto $P$
     if $s \in S_\top \cup S_\bot$ then break endif
     [cyclic: if $s$ is $\epsilon$-consistent then break endif]
     for all $a$ and $t$ where $P(s, a, t) > 0$ do
        if $t \notin \Theta$ then $Initialize(t)$ endif
     endfor
     update $V^U(s), \pi^U(s), V^L(s), \pi^L(s)$
     $s$ := sample $t$ according to $P(s, \pi^U(s), t)$
  endwhile
  while $P$ not empty do
     pop $s$ from $P$
     [acyclic: if $\neg$ $CheckSolved(s, 0)$ then break endif]
     [cyclic: if $\neg$ $CheckSolved(s, \epsilon)$ 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 (2003). 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.

3.3 LRTDP

For LRTDP, we consider the general case, including cyclic state spaces. Figure 2 shows pseudo-code. The main change consists in maintaining a lower bound in addition to the upper (optimistic) bound maintained by the original algorithm, and testing the exact same early termination criteria as in GoalProb-AO*. The latter is valid because, as before, $V^U(s)$ and $V^L(s)$ are upper and lower 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 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. [4]

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.

We will usually omit the "GoalProb-" in algorithm names.

[4]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.

Keep in mind though that our variants of AO* and LRTDP differ from the standard versions, in particular in early termination which depends on the objective MaxProb, AtLeastProb, ApproxProb. To study early termination capabilities, for X $\in$ {AO*, LRTDP} we will consider variants X$|_\mathrm{U}$ and X$|_\mathrm{L}$, maintaining only V^U respectively only V^L. Early termination criteria involving the non-maintained bound are disabled. For X$|_\mathrm{U}$, this leaves just the negative criterion $V^U(I) < \theta$ in AtLeastProb; X$|_\mathrm{L}$ still has positive criteria. We write X$|_\mathrm{LU}$ to make explicit that both bounds are used. For AO$^*|_\mathrm{L}$, 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. We do not consider a variant LRTDP$|_\mathrm{L}$ as LRTDP without an upper bound does not make sense.

3.4 FRET

As previously hinted, Kolobov et al.'s (2011) FRET performs an iteration of complete searches. Within each iteration, LRTDP or some other heuristic search algorithm finds a fixed-point upper bound V^U (more precisely, an ϵ-consistent V^U). In between iterations, FRET runs a *trap elimination* step, which finds traps – non-goal strongly connected components – in the greedy-policy graph with respect to V^U, and forces the next search iteration to not include these traps.

More precisely, FRET terminates if the greedy-policy graph does not contain a trap and otherwise removes every trap T in that graph by collapsing T's states into a single state s_T. The incoming transitions of s_T are those incoming to any state of the trap, and whose outgoing transitions are those transitions of trap states exiting the trap. This transformation obviously prevents T from occuring again in later iterations. It does not affect V^* as by definition the trap states have identical V^* values. As there is only a finite number of possible traps in the state space, FRET eventually finds a V^U whose greedy-policy graph does not contain any traps. From that graph, a V^U-greedy policy is extracted, which does not contain traps and hence is proper. In this way, FRET guarantees convergence to optimality. (This is the correctness argument given by Kolobov in his dissertation (Kolobov 2013).)

Now, in Kolobov et al.'s formulation, trap elimination considers the graph induced by *all* actions greedy with respect to V^U. We will refer to this design as *FRET-V^U*. Our alternative design, *FRET-π^U*, instead considers only the graph induced by the actions π^U actually selected into the current greedy policy. It is easy to see that this alternative method is still correct. The arguments above remain intact exactly as stated, the only difference being that we terminate based on whether the current V^U-greedy policy π^U is proper.

FRET-V^U potentially eliminates more traps in each iteration, and may hence require less iterations. Yet each trap elimination step may be much more costly. In particular, in goal probability analysis, V^U often is 1 almost everywhere in the first step, and the graph considered by FRET-V^U is almost the entire reachable state space. As we shall see, FRET-π^U clearly outperforms FRET-V^U on the standard benchmarks.

4 State-Space Reduction via Determinized Bisimulation

Bisimulation is a known method to reduce state space size in MDPs/probabilistic planning (e. g. (Dean and 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 (Helmert et al. 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 $B_i, B_j \in P$, every action a, and every $s, t \in B_i$, the following two properties are satisfied (Dean and 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 (1997) 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 at the level of the block states B_i.

Now, denote by Π^{det} the *all-outcomes determinization* of Π(e. g. (Yoon, Fern, and Givan 2007; Little and 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 state set S so that, for every $B_i, B_j \in P$, 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.

To compute a determinized bisimulation for Π, one can use merge-and-shrink with the widely employed shrinking strategies based on bisimulation. As we shall see in the experiments, this often still incurs a prohibitive overhead, but also often is feasible and leads to substantial state space size reductions. In some cases, it results in tremendous performance improvements.

5 Dead-End Pruning

Dead-ends are states s where $V^*(s) = 0$. One can treat such s exactly like lost states $S_\perp$ (except for setting $\pi^L(s) := *$). Apart from this pruning itself, for the heuristic search algorithms this provides a non-trivial initialization of V^U, typically leading to additional search reductions.

Kolobov et al. (2011) employ SixthSense (Kolobov, Mausam, and Weld 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. This works especially well in limited-budget planning, where we can use lower bounds on determinized remaining cost to detect states with insufficient remaining budget. Note that this is natural and effective using admissible remaining-cost estimators, yet would be impractical using an actual planner (which would need to be optimal and thus prohibitively slow). For the general 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, and 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 unsolvable states, 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 $b(s) < h(s)$, 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 and Domshlak 2009), (c) several variants of merge-and-shrink heuristics, and (d) $h^{\max}$ (Bonet and 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 to adopt the problem reformulation by Domshlak and Mirkis (2015) for oversubscription planning, which reduces the budget b using landmarks and in exchange allows to traverse yet non-used landmarks at a reduced cost during search. Somewhat surprisingly, however, pruning states whose reduced budget is < 0 is equivalent to the much simpler method pruning states whose heuristic (a) exceeds the 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 A. 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 to apply 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 $b - |\vec{a}| < |L'|$. The latter condition, however, is exactly the pruning condition using the simple method (a) instead.

6 Node Selection Strategies

In both GoalProb-AO* and GoalProb-LRTDP, 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 the outcome-state sampling during trials (LRTDP), respectively 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 policy 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.

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 standard use of AO* and LRTDP. We use 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 for LRTDP (2003). There is no bias on outcome states in AO* (an open outcome state is selected arbitrarily), and the bias 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.[5] Precisely, for action selection tie-breaking (1), from those actions 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, Likhachev, and Gordon 2005), we experiment with a *gap-bias* strategy, biasing

[5]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, and Smith 2006)). This is compelling in theory because, then, a perfect bisimulation-based heuristic as per Nissim et al. (2011) corresponds to the exact goal probability of the best outcome sequence from a state. Unfortunately, as we show in the next section, computing such a merge-and-shrink heuristic is often infeasible.

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 and Nebel 2001; Helmert 2006; Richter and 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. In what follows, we first say a few words on our implementation, and describe the benchmarks. We then summarize our experiments on acyclic benchmarks (where FRET is not needed), then the ones for cyclic benchmarks (where FRET is needed).

7.1 Experiments Setup

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 budget limit.

Naturally, 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.)

Given the scant prior work on optimal goal probability analysis, as explained in the introduction, 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 pruning, which was SixthSense in the only prior work, by Kolobov et al. (2011), using such pruning. 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 includes a detailed comparison of our results with those by Kolobov et al., on IPPC ExplodingBlocks which is the only domain considered by Kolobov et al.

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 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.[6] 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, we modified the POMDP generator by Sarraute et al. (2012), which itself is based on a test scenario used at Core Security (`http://www.coresecurity.com/`). 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. We modified the generator to output PPDDL encodings of Hoffmann's (2015) *attack-asset MDP* pentesting models. Sarraute et al.'s POMDP model and solver (SARSOP (Kurniawati, Hsu, and Lee 2008), which does not guarantee optimality) scale to $H = 6, E = 10$.[7] 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 task Π, except the pentesting ones, 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 opti-

[6] 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.

[7] 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.

mal 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 6 search algorithm variants, 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 (BS) reduced state space only for VI, as the cases where the state space reduction succeeds are easily solved by that simplest search algorithm. Given BS, we do not require any dead-end pruning because all dead-ends are already removed from the reduced state space.

Overall, this yields 217 different possible algorithm configurations. We do not actually test all these configurations, however, as not all of them are interesting. We instead organize our experiment in terms of three parts (1)–(3), each focusing on a particular issue of interest. Consider Table 1, which gives an overview of the configurations considered in each experiment. The design of the experiments is as follows:

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

    We omit here $AO^*|_{LU}$ and $LRTDP|_{LU}$, 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 make 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.

Experiment	Search Algorithm	Pruning	Node selection	# Configs
(1) MaxProb search & pruning	VI, $AO^*|_L$, $AO^*|_U$, $LRTDP|_U$, VI on BS	ALL (4)	default	20
(2) AtLeastProb & ApproxProb parameters	VI, $AO^*|_L$, $AO^*|_U$, $AO^*|_{LU}$, $LRTDP|_U$, $LRTDP|_{LU}$, VI on BS	LM-cut	default	14
(3) AtLeastProb & ApproxProb node selection	VI, $AO^*|_L$, $AO^*|_U$, $AO^*|_{LU}$, $LRTDP|_U$, $LRTDP|_{LU}$, VI on BS	LM-cut	ALL (1, 4, 4, 5, 3, 4, and 1 respectively)	44

Table 1: 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).

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 an illustration of typical anytime behavior.

(1) Search Algorithms & Pruning Methods in MaxProb Table 2 shows coverage data, i. e., the number of benchmark tasks solved within the given time/memory limits.

		VI				$AO^*|_L$				$AO^*|_U$				$LRTDP|_U$				VI
		–	LM	M&S		–	LM	M&S		–	LM	M&S		–	LM	M&S		on
Domain	#			N	∞			N	∞			N	∞			N	∞	BS
IPPC Benchmarks																		
TriaTire	10	4	4	4	4	4	4	4	4	**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
Boxworl-b	18	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	52
Elevator-b	90	71	82	72	33	70	82	72	33	65	77	67	33	79	**86**	79	33	33
ExpBloc-b	84	32	46	38	37	32	46	38	37	39	57	39	37	38	**65**	39	37	37
Random-b	60	27	33	35	33	27	33	35	33	35	**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**	**36**
Tirewor-b	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	**60**
Zenotra-b	36	15	16	16	**18**	15	16	16	**18**	14	16	16	17	15	17	16	17	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	**51**
Rovers-b	60	23	39	31	40	23	38	31	40	23	44	33	45	25	46	35	46	**50**
TPP-b	60	18	35	25	25	16	35	24	24	15	37	26	22	19	**38**	27	25	26
Pentesting Benchmarks																		
Pentest	15	**9**	**9**	**9**	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8	**9**	**9**	**9**	8	8
Pentest-b	90	57	**63**	62	37	57	**63**	62	37	57	**63**	**63**	37	57	**63**	**63**	37	37
$\sum$	925	546	658	627	533	543	656	625	531	559	693	641	546	581	**718**	661	555	564

Table 2: Acyclic planning. MaxProb coverage (number of tasks solved within time & memory limits). Best values in **boldface**. 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 BS": VI run on reduced (bisimulated) state space.

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 as a reduced MDP state space ("VI on BS"), rather than only for dead-end pruning. An extreme example is Triangle-Tireworld. Far beyond the standard benchmarks in Table 2 (triangle-side length 20), VI on BS 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, Muise, and McIlraith 2016), which scales to side length 70 on the original domain,[8] and is optimal only for goal probability 1, i. e., in the presence of strong cyclic plans.

Of the search algorithms, $AO^*|_L$ 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 $AO^*|_L$ is dominated by VI (this changes for AtLeastProb, Figures 4 (a) and 5 below). All failures of VI are due to memory or runtime exhaustion while building the reachable state space. $LRTDP|_U$ clearly outperforms $AO^*|_U$, presumably because it tends to find absorbing states more quickly.

To gauge the efficiency of heuristic search vs. blind search on MaxProb, compare $LRTDP|_U$ vs. VI in Table 2. Contrary to the intuition that a good initial goal probability estimator is required for heuristic search to be useful, $LRTDP|_U$ 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, $LRTDP|_U$ dominates VI throughout, and improves coverage in 8 of the 16 domains.

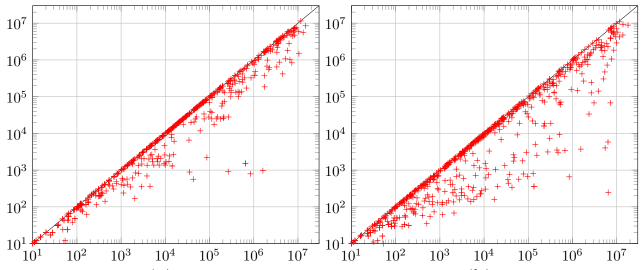


Figure 3: Acyclic planning. Number of states visited, for VI (x) vs. $LRTDP|_U$ (y), with no pruning (a) respectively LM-cut pruning (b).

Figure 3 sheds additional light on this by comparing the respective search space sizes directly. The non-trivial initialization using LM-cut clearly helps. But even without it, gains of around 1 order of magnitude occur frequently, and larger gains (up to 3 orders of magnitude) occur in rare cases. 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

		VI				$AO^*\|_U$				$LRTDP\|_U$			
		–	LM	M&S		–	LM	M&S		–	LM	M&S	
Domain	#			N	∞			N	∞			N	∞
IPPC Benchmarks													
TriaTire	1	3.5	7.4	4.1	4.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0
	1					0.1	0.3	1.0	1.0	0.2	0.5	1.0	1.0
IPPC Benchmarks with Budget Limit													
Blocksw-b	18	0.1	0.6	2.5	2.3	1.8	0.8	3.0	2.8	0.2	0.6	2.3	2.4
Drive-b	20	0.0	0.2	6.9	14.0	0.1	0.2	8.0	15.4	0.0	0.2	7.1	14.2
ONLY-H	1					0.0	0.1	1.1	1.1	0.0	0.1	0.9	1.0
Elevator-b	12	0.1	0.1	1.8	4.1	0.0	0.0	2.2	4.5	0.0	0.0	1.8	4.1
ONLY-H	3					0.0	0.0	1.0	0.9	0.0	0.0	0.8	0.7
ExpBloc-b	18	6.0	1.0	15.5	7.5	1.6	0.0	16.0	8.0	0.8	0.1	14.2	7.1
NON-TRIVIAL	7	25.3	4.8	36.2	45.7	11.5	0.1	33.0	45.9	4.0	0.1	29.7	42.1
ONLY-H	3					29.3	0.1	40.9	40.4	21.4	0.1	30.7	34.9
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
NON-TRIVIAL	4	13.9	10.1	39.2	43.2	3.0	0.9	44.4	49.9	1.4	0.8	36.3	43.2
ONLY-H	3					9.2	1.9	11.8	12.3	9.2	1.8	10.1	10.6
RecTire-b	18	9.0	19.4	1.2	1.2	73.6	20.2	1.3	1.3	43.1	17.6	1.3	1.3
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
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
NON-TRIVIAL	6	27.7	5.9	3.2	3.3	31.3	2.4	2.1	2.0	14.7	2.0	1.7	1.6
ONLY-H	3					6.3	0.3	0.5	0.5	3.5	0.2	0.3	0.3
Zenotra-b	14	2.7	4.9	15.0	9.0	56.0	5.7	18.9	11.8	13.0	4.3	15.9	9.2
NON-TRIVIAL	10	5.6	16.5	27.0	13.5	163.4	19.3	37.2	18.7	25.3	13.6	30.1	14.5
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
ONLY-H	1					1623.4	8.9	0.6	0.6	158.8	7.8	0.5	0.4
Rovers-b	21	9.7	2.3	11.8	17.0	96.2	2.3	16.5	21.7	12.8	2.0	11.6	16.1
NON-TRIVIAL	13	20.9	12.9	20.2	33.7	236.6	12.0	33.3	45.1	24.9	9.7	19.5	30.3
ONLY-H	3					157.6	1.9	13.7	22.5	27.9	1.9	9.7	19.4
TPP-b	9	8.5	1.6	14.9	69.8	63.2	1.3	24.6	70.2	12.7	1.3	16.0	69.1
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
Pentesting Benchmarks													
Pentest	3	0.9	0.7	3.2	4.4	5.9	2.3	5.7	6.5	3.8	3.0	5.8	6.3
NON-TRIVIAL	1	2.7	2.0	6.5	17.2	23.0	8.1	20.6	23.8	15.0	10.4	16.6	24.4
Pentest-b	28	0.0	0.0	6.6	16.5	0.5	0.0	8.2	19.8	0.0	0.0	7.3	18.2
NON-TRIVIAL	5	3.2	2.2	10.1	92.7	16.0	4.5	15.0	108.9	8.5	5.2	15.2	107.6
ONLY-H	1					0.1	0.0	0.9	0.9	0.1	0.0	1.0	1.0

Table 3: Acyclic planning. MaxProb geometric mean runtime (in CPU seconds). "#" 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 $AO^*|_U$ and $LRTDP|_U$ but not solved by VI. Rows with empty instance basis are skipped.

LRTDP already benefits from a smaller state space, and the impact of heuristic search remains unclear.

Tables 3 and 4 complement the above with per-domain mean runtime and visited-states data, across search algorithms and heuristic functions. Data for $AO^*|_L$ is not shown as its coverage is dominated by VI (cf. Table 2), 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 basic observation from this data is that the main advantage of heuristic search here lies in the ability to solve larger instances than blind search (VI). 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–

[8]We could not run the limited-budget domain as Prob-PRP does not natively support a budget, and hard-coding the budget into PPDDL resulted in encodings too large to pre-process.

		VI				$AO^*\|_U$				$LRTDP\|_U$			
		–	LM	M&S		–	LM	M&S		–	LM	M&S	
Domain	#			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
	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
	1					0.5	0.5	0.5	0.5	5.5	2.2	2.2	2.2
IPPC Benchmarks with Budget Limit													
Blocksw-b	18	12.7	5.8	2.8	2.8	12.0	5.2	2.5	2.5	12.3	5.3	2.5	2.5
Drive-b	20	4.2	2.4	1.8	1.2	4.2	2.2	1.6	1.0	4.0	2.1	1.5	1.0
ONLY-H	1					2.3	1.3	0.8	0.8	2.2	1.2	0.7	0.7
Elevator-b	12	12.8	3.9	7.2	3.0	3.3	0.3	0.4	0.1	3.6	0.3	0.4	0.1
ONLY-H	3					1.5	0.2	0.1	0.1	1.6	0.2	0.1	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
NON-TRIVIAL	7	4.1K	213.2	859.9	179.0	603.1	3.6	133.6	2.6	588.0	4.0	106.8	3.2
ONLY-H	3					2.6K	1.5	17.2	1.1	3.1K	2.1	21.3	1.6
Random-b	21	4.5	1.7	1.7	1.7	1.9	0.8	0.8	0.8	2.0	0.8	0.8	0.8
NON-TRIVIAL	4	1.0K	130.0	127.4	127.4	42.6	6.4	6.4	6.4	43.7	6.4	6.4	6.4
ONLY-H	3					762.1	5.4	1.5	1.5	799.6	5.4	1.5	1.5
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
NON-TRIVIAL	12	81.9	8.9	2.4	2.4	81.5	8.2	1.9	1.9	81.6	8.3	2.0	2.0
TriaTire-b	17	1.4K	6.4	6.4	6.4	898.6	3.0	3.0	3.0	896.2	2.9	2.9	2.9
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
ONLY-H	3					422.9	2.5	2.5	2.5	370.5	2.0	2.0	2.0
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
NON-TRIVIAL	10	967.4	104.4	164.6	64.0	967.1	102.9	161.1	62.3	478.1	75.3	114.9	45.8
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
ONLY-H	1					12.4K	122.4	14.1	14.1	12.7K	122.3	16.4	16.4
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
NON-TRIVIAL	13	2.2K	290.1	512.6	137.6	1.1K	176.0	281.4	65.9	1.6K	190.0	366.2	76.3
ONLY-H	3					1.3K	34.0	55.2	24.1	2.0K	38.6	72.8	27.8
TPP-b	9	1.1K	49.6	265.4	10.9	660.9	33.0	183.3	5.7	897.2	38.5	220.7	7.6
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
Pentesting Benchmarks													
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
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
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
NON-TRIVIAL	5	238.1	165.1	169.2	165.1	237.2	165.1	169.0	165.1	238.1	165.1	169.1	165.1
ONLY-H	1					4.7	3.3	3.3	3.3	4.8	3.3	3.3	3.3

Table 4: Acyclic planning. MaxProb geometric mean search space size (number of different states visited before termination) in multiples of 1000. Same setup and presentation as in Table 3.

b. The search space reductions are much more pronounced. In particular, even without dead-end detection i. e. with a trivial initialization of the admissible bound, search spaces are reduced in all domains except RectangleTireworld and Pentest.

$LRTDP|_U$ 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 table, the geometric mean of the number of updates done in $AO^*|_U$ is about 4 times higher than that in $LRTDP|_U$.

Regarding the impact of dead-end pruning, the gains for VI are typically moderate. The gains for heuristic search are much more pronounced, thanks to the stronger heuristic function initialization. Especially $AO^*|_U$ benefits a lot. $LRTDP|_U$ benefits as well, but to a smaller extent, partly because it is already more effective in the first place. Comparing across different dead-end pruning methods, although M&S with $N = \infty$ clearly leads to the largest search space reductions, 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 about on par, though LM-cut has a slight edge in terms of runtime.

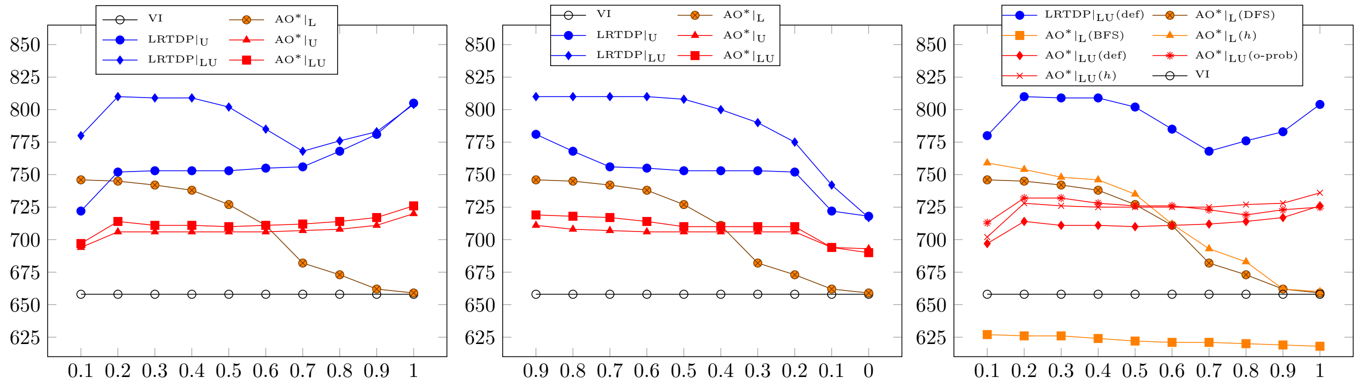
(2) AtLeastProb and ApproxProb Parameter Analysis We now turn to the weaker objectives, AtLeastProb and ApproxProb. We fix LM-cut for the (almost always most effective) dead-end pruning. We examine the power of early termination for different search algorithms and node selection strategies. This is best 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. Consider Figure 4.

For AtLeastProb (Figure 4 (a)), in the interesting region of benchmark instances not feasible for VI yet sometimes feasible for the other search algorithms,one clear feature is again the superiority of LRTDP over AO*. There is now the striking exception of $AO^*|_L$, however, which for small values of θ approaches (and in one case, surpasses) the performance of LRTDP. 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 $AO^*|_{LU}$. As we shall see below (Figure 5), 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 4 (a), i. e., significantly worse performance for $\theta = 0.1$ than for $\theta = 0.2$, is an outlier due to the Pentest domains (without them, $AO^*|_{LU}$ and $LRTDP|_{LU}$ exhibit a strict easy-hard-easy pattern). In contrast to typical probabilistic planning scenarios, in penetration testing the goal probability – the chances 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 4 (b)), smaller values of δ consistently result in worse performance. We see again the superiority of LRTDP over AO*, with a similar though not as pronounced exception for $AO^*|_L$ in δ regions allowing aggressive early termination. We also see again the superiority of algorithms using both bounds over those that don't.

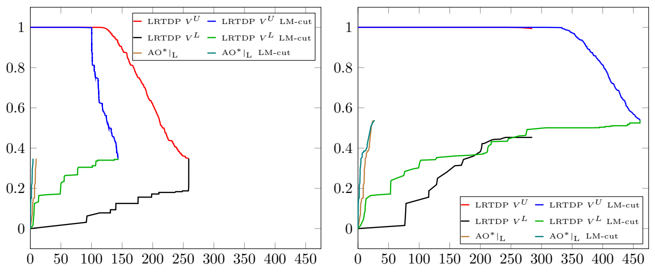
(3) Node Selection Strategies Figure 4 (c) examines the effect of different node selection strategies in AtLeastProb (the relative performance of node selection strategies is the same in ApproxProb, so we do not include a separate figure for that). For readability, we show only the most competitive base algorithms, $AO^*|_L$, $AO^*|_{LU}$, and $LRTDP|_{LU}$ (as well as the VI baseline). For LRTDP, we show only default node selection, which consistently works a little better than the alternatives. For $AO^*|_L$, we see that the depth-first strategy is important (and way beyond breadth-first, which does worse than VI). The h-bias strategy is generally on par with depth-first. For $AO^*|_{LU}$, the main observation is that the most-prob-outcome bias is helpful, improving over the default strategy except for high values of θ. 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



(a) (b) (c)

Figure 4: Acyclic planning. Total coverage for AtLeastProb as a function of θ in (a) and (c), for ApproxProb as a function of δ in (b). Node selection varies in (c), default used in (a) and (b). All configurations use LM-cut dead-end pruning.

mainly cause runtime overhead).



(a) (b)

Figure 5: Acyclic planning. Anytime behavior in LRTDP$|_{LU}$ (V^U and V^L) and AO*$|_L$ (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).

An Illustration of Typical Anytime Behavior To conclude our discussion of acyclic planning, Figure 5 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$|_{LU}$ and AO*$|_L$ (using default node selection because the alternatives are not beneficial for these algorithms). The benefit of LM-cut pruning is evident. Observe that AO*$|_L$ 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*$|_L$ and LRTDP, omitting those where both took < 1 second, in 56% of cases AO*$|_L$ finds an optimal policy faster than LRTDP. On (geometric) average, AO*$|_L$ takes 66% of the time taken by LRTDP for this purpose. On the downside, unless $V^*(I) \geq \theta$, AO*$|_L$ must explore the entire state space. Its runs in Figure 5 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 5 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 (cf. $C = 1.8$ vs. $C = 1.4$ in Figure 5). 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, 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 as (like we have seen before) in LRTDP these do not bring an advantage over the default strategy. We use the deterministic-bisimulation (BS) 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 BS we do not require any dead-end pruning.

Overall, this yields 55 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) and (2) as before. As node selection strategies are not relevant here, we do not have the previous part (3). Table 5 gives an overview.

(1) Search Algorithms & Pruning Methods in MaxProb Table 6 shows coverage data. Most strikingly, 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. Figure 6 compares the search space sizes. The non-trivial initialization using

Experiment	FRET variant	Search Algorithm	Pruning	# Configs
(1) MaxProb search & pruning	–, FRET-V^U, FRET-π^U	VI, LRTDP$\|_{\mathrm{U}}$	ALL (4), BS	15
(2) AtLeastProb & ApproxProb parameters	–, FRET-V^U, FRET-π^U	VI, LRTDP$\|_{\mathrm{U}}$, LRTDP$\|_{\mathrm{LU}}$	$h^{\max}$	10

Table 5: Overview of algorithms tested on cyclic problems, Section 7.3. Note that VI does not require (and is this 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).

		VI					FRET-V^U					FRET-π^U				
		–	$h^{\max}$	M&S		on	–	$h^{\max}$	M&S		on	–	$h^{\max}$	M&S		on
Domain	#			N	∞	BS			N	∞	BS			N	∞	BS
IPPC Benchmarks																
Blocksworld	15	**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
Drive	15	**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
ExplodingBlocks	15	4	6	4	4	4	4	6	4	4	4	5	**14**	5	4	4
Random	15	0	0	0	0	0	0	0	0	0	0	**4**	**4**	0	0	0
RectangleTireworld	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	**15**	**15**	**15**	12	11
Zenotravel	15	**3**	**3**	**3**	1	0	**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**	4	4	4	4	1
Rovers	10	5	5	5	5	**9**	5	5	5	5	**9**	**9**	**9**	**9**	8	**9**
TPP	10	6	6	6	6	6	6	6	6	6	6	**8**	**8**	**8**	6	6
$\sum$	164	81	83	81	60	64	81	84	81	60	65	96	**105**	92	64	61

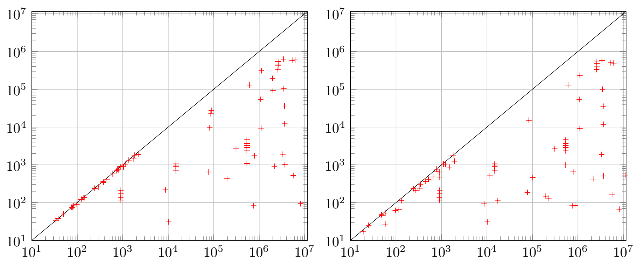
Table 6: Cyclic planning. MaxProb coverage. Best values in **boldface**. FRET-V^U is as per Kolobov et al. (2011), FRET-π^U is our modified version. Both use LRTDP$|_{\mathrm{U}}$. Dead-end pruning variants: "–" none, else based on heuristic value ∞, for $h^{\max}$ respectively merge-and-shrink ("N" size bound $N = 100k$, "∞" no size bound). "on BS": run on reduced (bisimulated) state space.

$h^{\max}$ is useful, but gains of up to 3 orders of magnitude are possible even without it.

Running the search on a deterministic-bismulation state space is less effective on the cyclic benchmarks in Table 6, than on the acyclic ones in Table 2. It gives a clear advantage only in Rovers.

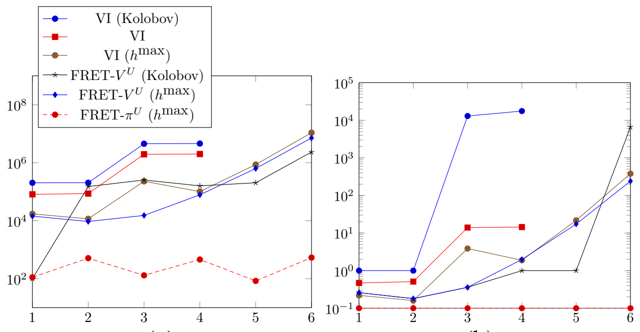
Table 7 gives per-domain mean runtime and visited-states data, across search algorithms and heuristic functions. In contrast to the acyclic case, there is only one domain where dead end pruning is able to reduce the search space substantially: ExplodingBlocks. In the other domains, there either is no reduction or a minor/moderate one only. Among the tested search algorithms, strikingly, FRET-π^U consistently visits the smallest number of states, by far. 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.

Kolobov et al. (2011) experimented only on ExplodingBlocks, and only ran VI with no pruning vs. FRET-V^U with pruning based on SixthSense (Kolobov, Mausam, and Weld 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 $h^{\max}$ here. Figure 7 compares the results in more detail,



(a) (b)

Figure 6: Cyclic planning. Number of states visited, for VI (x) vs. FRET-π^U (y), with no pruning (a) respectively $h^{\max}$ pruning (b).



(a) (b)

Figure 7: 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.

showing the number of states visited and the total runtime in terms of plots over IPPC instance index as done by Kolobov et al.

Consider first (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 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 $h^{\max}$ – which is hardly surprising seeing as the information sources in SixthSense are full determinized planning as well as h^2 (Graphplan) based tests. On the other hand, these information sources are much more time-intensive than $h^{\max}$, which presumably is the reason for the runtime picture in

Domain	#	VI				FRET-V^U				FRET-π^U			
		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S	
				N	∞			N	∞			N	∞
IPPC Benchmarks													
Blocksworld	4	0.0	0.0	2.8	2.7	0.0	0.1	2.7	2.8	0.1	0.1	3.0	2.8
Drive	1	0.0	0.0	6.0	42.8	0.0	0.0	5.8	33.1	0.0	0.0	6.1	42.8
Elevators	5	0.0	0.0	2.2	1.7	0.0	0.0	2.2	1.8	0.0	0.0	2.2	1.9
ExplodingBlocks	4	2.6	0.7	19.3	18.1	15.9	0.4	31.5	17.7	15.7	0.0	28.8	17.5
Tireworld	8	7.3	10.9	14.0	35.0	60.0	55.3	60.2	86.4	0.0	0.0	3.8	24.7
Zenotravel	1	55.7	49.3	96.3	283.8	7.0	12.6	54.3	241.1	0.2	0.2	43.5	227.4
Probabilistic Resource-Constrained Benchmarks													
NoMystery	4	21.5	29.8	29.1	69.4	133.9	127.0	141.4	166.7	627.3	582.4	676.4	618.7
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
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

Domain	#	VI				FRET-V^U				FRET-π^U			
		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S		–	$h^{\max}$	M&S	
				N	∞			N	∞			N	∞
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
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
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
ExplodingBlocks	4	408.5	46.5	252.3	39.9	408.5	20.1	242.2	16.9	44.3	0.2	14.0	0.1
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
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
Probabilistic Resource-Constrained Benchmarks													
NoMystery	4	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	2.6K	433.0	430.8	433.0	430.8
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
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

Table 7: Cyclic planning. Top: MaxProb geometric mean runtime (in CPU seconds). Bottom: MaxProb geometric mean search space size (number of different states visited before termination) 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. Rows with empty instance basis are skipped.

(b), which 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.

(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 8 shows the data for default node selection in AtLeastProb. By FRET$|_\mathrm{U}$ respectively FRET$|_\mathrm{LU}$, we refer to FRET using LRTDP$|_\mathrm{U}$ respectively LRTDP$|_\mathrm{LU}$.

For FRET-V^U, the behavior is similar to Figure 4 (a), FRET$|_\mathrm{LU}$-V^U exhibiting an easy-hard-easy pattern due to the advantages of early termination. For FRET-π^U, though, the curves are flat over θ. 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 9 exemplifies this. The easy-hard-easy pattern would thus emerge for smaller runtime/memory limits.[9]

[9]Figure 9 (b) considers the largest instance feasible when using $h^{\max}$ pruning. Figure 9 (a) considers the second-largest instance feasible without pruning: on the largest one, 05, the maximum goal probability is 1 so the anytime curve for V^U is not interesting. We remark that, in ExplodingBlocks 04, with $h^{\max}$ pruning V^U and V^L converge instantly. In ExplodingBlocks 15, when not using pruning, V^U and V^L remain 0 respectively 1 throughout.

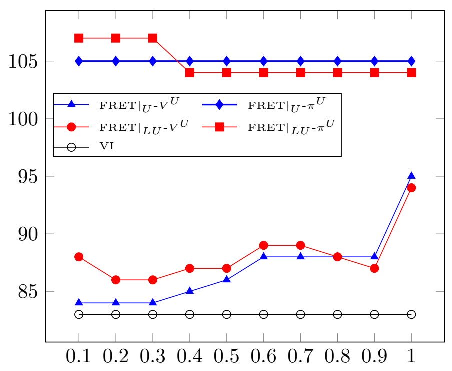


Figure 8: Cyclic planning. AtLeastProb total coverage as a function of θ, using $h^{\max}$ dead-end pruning and default node selection.

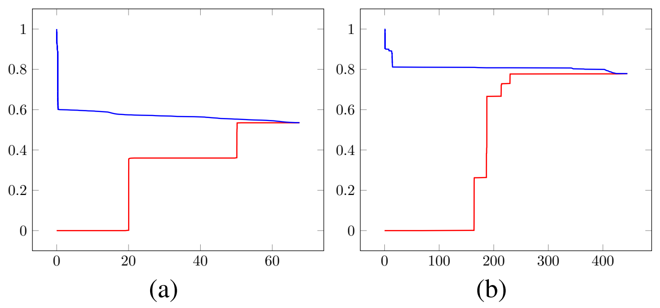


Figure 9: Cyclic planning. Anytime behavior of FRET-π^U with LRTDP$|_\mathrm{LU}$, (a) without pruning for ExplodingBlocks instance 04, and (b) with $h^{\max}$ pruning for instance 15.

Figure 10 shows coverage data for default node selection in ApproxProb, as a function of δ. The behavior is qualitatively similar to that observed for AtLeastProb in Figure 8.

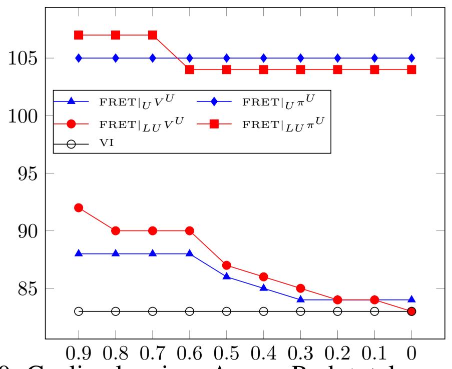


Figure 10: Cyclic planning. ApproxProb total coverage as a function of δ, using $h^{\max}$ dead-end pruning and default node selection.

8 Conclusion

Optimal goal probability analysis is a notoriously hard problem, to the extent that the amount of work addressing it is limited. We clarified the empirical state of the art, and substantially improved it through a novel variant of FRET and through a novel state-space reduction method. We showed that there are opportunities arising from naturally acyclic problems, and from early termination on criteria weaker than maximum goal probability. We hope that this will inspire renewed interest in this important problem. Promising future directions include advanced admissible goal probability estimators, e. g. from abstractions interpreted as bounded-parameter MDPs (Givan, Leach, and Dean 2000); hybrids of heuristic search with Monte-Carlo tree search, geared at good anytime behavior and thus early termination; and the exploitation of goal probability monotonicity as a function of remaining budget. Simulated pentesting is an application worth algorithms research in it own right. Partial-order reduction appears especially promising there.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1. 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.

References

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

Barto, A. G.; Bradtke, S. J.; and 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.

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

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

Bonet, B., and 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., and Smith, S., eds., *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 142–151. Ambleside, UK: Morgan Kaufmann.

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

Camacho, A.; Muise, C.; and 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.; and Sanner, S., eds., *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*. AAAI Press.

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

Coles, A. J.; Coles, A.; Fox, M.; and 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. 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)*, 252–257. Montpellier, France: IOS Press.

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

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

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

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

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

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

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

Helmert, M.; Haslum, P.; Hoffmann, J.; and 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).

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

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

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

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

Jimenez, S.; Coles, A.; and 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., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, 1728–1733. Pasadena, California, USA: Morgan Kaufmann.

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

Kolobov, A.; Mausam; and Weld, D. S. 2010. Sixthsense: Fast and reliable recognition of dead ends in MDPs. In Fox, M., and 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; and Weld, D. S. 2012. A theory of goal-oriented MDPs with dead ends. In de Freitas, N., and Murphy, K. P., eds., *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI'12)*, 438–447. Catalina Island, CA, USA: AUAI Press.

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

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

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

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

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

McMahan, H. B.; Likhachev, M.; and 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.; and 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*. Elsevier and MIT Press. 1201–1242.

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

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

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

Nissim, R.; Hoffmann, J.; and 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)*, 1983–1990. AAAI Press/IJCAI.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 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`.

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

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

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

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

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

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

A 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' 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 bookkeeping 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 $b'(s) < 0$.[10] 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 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 does this kind of book-keeping, and uses the admissible heuristic value $h(L(s))$.

If one were to apply Domshlak and Mirkis' 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

[10]Domshlak and Mirkis 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.

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' 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.