# All that Glitters is not Gold: Using Landmarks for Reward Shaping in FPG

Olivier Buffet, Joerg Hoffmann

# All that Glitters is not Gold: Using Landmarks for Reward Shaping in FPG

**Olivier Buffet** and **Jörg Hoffmann**
INRIA, France
olivier.buffet|joerg.hoffmann@inria.fr

## Abstract

Landmarks are facts that must be true at some point in any plan. It has recently been proposed in classical planning to use landmarks for the automatic generation of heuristic functions. We herein apply this idea in probabilistic planning. We focus on the FPG tool, which derives a factored policy based on learning from samples into the state space. The rationale is that FPG's performance can be improved significantly by a trivial heuristic that counts the number of false goals; landmarks provide much better estimates at little overhead cost.

We devise improved versions of the classical landmarks heuristic, including a Markovian one that, unlike previous ones, does not depend on the state history. As done previously in FPG for the goal counting, we use the heuristics for reward shaping: the planner gets a positive reward when improving the heuristic value. Based on previous work, we argue that such shaping is policy invariant for Markovian heuristics.

Our empirical results confirm that the landmarks heuristics are almost as fast as the goal counting, while delivering much more accurate estimates for initial states. In spite of this, overall planner performance is almost never improved. We discuss some intuitions as to why that is so.

## Introduction

Landmarks are facts that must be true at some point in any plan. It has been originally proposed to detect landmarks, and ordering constraints between them, in a pre-process, and then use this information for problem decomposition (Hoffmann, Porteous, and Sebastia 2004). More recently, it has been proposed in classical planning to use landmarks for the automatic generation of heuristic functions (Richter, Helmert, and Westphal 2008; Karpas and Domshlak 2009). The basic idea is to count the number of yet un-achieved landmarks, enforcing, where appropriate, achievement in the order dictated by the ordering constraints. We herein apply these (classical) heuristic functions, for the first time, in probabilistic planning. We focus on the FPG tool (Buffet and Aberdeen 2009) that derives a factored policy based on learning from samples into the state space. The rationale behind this research is that FPG's performance can be improved significantly by a trivial heuristic that counts the number of false goals; landmarks provide much better estimates at little overhead cost.

To illustrate our rationale, consider Table 1, where we compare the success rates for policies computed by FPG, in Blocksworld ("BW", IPPC 2006) and Exploding-Blocksworld ("EBW", IPPC 2006), without vs. with the goal counting heuristic (all other settings being the same). Clearly, this trivial heuristic has a huge beneficial impact on performance. It also has limitations, visible in the results on BW12 and EBW10. Our initial hope in this work was to be able to scale to these cases when exchanging the goal counting with the much more informed counting of landmarks.

| | BW8 | BW12 | EBW5 | EBW8 | EBW10 |
|---|---|---|---|---|---|
| FPG | 9 | 0 | 51 | 0 | 0 |
| FPG+$h^{\mathsf{G}}$ | 100 | 4 | 42 | 50 | 0 |

Table 1: Success rate of FPG policies, computed without vs. with goal counting heuristic.

Like previously done in FPG for the goal counting, we use the heuristics for *reward shaping* (Dorigo and Colombetti 1994; Matarić 1994). The planner receives a positive reward when improving the heuristic value, hence guiding the learning towards smaller values. Reward shaping is called *policy invariant* if it preserves the optimal policy of the problem at hand, i.e., the policy that maximizes expected reward. Previous work (Ng, Harada, and Russell 1999) has shown that heuristics-based reward shaping is policy invariant under the standard restrictive assumptions implying that expected reward is well-defined (using discounting or the existence of an absorbing state). These assumptions are not per se given in probabilistic planning – there may exist dead-end states from which the goal can't be reached – but can be achieved by inserting an artificial absorbing state; the reward function is designed to distinguish between successful and failed ways to reach that state. To the transformed problem, the results of (Ng, Harada, and Russell 1999) apply as before.

The heuristics we use are all classical, and we take the known approach of applying them to the all-outcomes determinization of the probabilistic planning task. We devise two new versions of the landmarks heuristic from (Richter, Helmert, and Westphal 2008). The first version makes improvements pertaining to an analysis of undesirable features of the original proposal, such as potential failure to recognize goal states. The second version, in difference to the previous versions of the landmarks heuristic, is Markovian, i.e., does not depend on the state history. This is of potential importance in our context because the results of (Ng, Harada, and Russell 1999) pertain only to this kind of heuristic.

Our experiments confirm that the landmarks heuristics are almost as fast as the goal counting, while delivering much more accurate estimates for initial states. In spite of this, overall FPG performance is almost never improved. We discuss some intuitions as to why that is so. We experiment also with the well-known relaxed plans heuristic (Hoffmann and Nebel 2001); this does not deliver good results either.

We next give some background on probabilistic planning and FPG. We then provide an overview of shaping methods and explain our strategy for FPG. We discuss in detail the different variants of the landmarks heuristic, before presenting the empirical results and concluding the paper.

## Background

### Probabilistic Planning

A probabilistic planning *domain* is defined by a finite set of boolean variables $\mathcal{B} = \{b_1, \ldots, b_n\}$ – a *state* $s \in \mathcal{S}$ being described by an assignment of these variables, and often represented as a vector $\mathbf{s}$ of 0s and 1s – and a finite set of *actions* $\mathcal{A} = \{a_1, \ldots, a_m\}$. An action $a$ can be executed if its *precondition* $pre(a)$ – a logic formula on $\mathcal{B}$ – is satisfied. If $a$ is executed, a probability distribution $\mathbb{P}(\cdot|a)$ is used to sample one of its $K$ *outcomes* $out_k(a)$. An outcome is a set of truth value assignments on $\mathcal{B}$ which is then applied to change the current state.

A probabilistic planning *task* $\mathcal{T}$ is defined by a planning domain, an initial state $s_0$ and a *goal* $G$ – a formula on $\mathcal{B}$ that needs to be satisfied. The aim is to find the plan that maximizes the probability of reaching the goal, and possibly minimizes the expected number of actions required. This takes the form of a policy $\mathbb{P}[a|\mathbf{s}]$ specifying the probability of picking action $a$ in state $s$. In the remainder of this section, we see how FPG solves this with RL.

### FPG

FPG addresses probabilistic planning as a Markov Decision Process (MDP): in its default version, a reward function $r$ is defined, taking value $r_{\text{succ}} = 1000$ in any goal (success) state, and 0 otherwise; a transition matrix $\mathbb{P}[s'|s, a]$ is naturally derived from the actions; the system resets to the initial state each time the goal is reached; and FPG tries to maximize the expected average reward. But rather than dynamic programming – which is costly when it comes to enumerating reachable states –, FPG computes gradients of a stochastic policy, implemented as a policy $\mathbb{P}[a|\mathbf{s}; \theta]$ depending on a parameter vector $\theta \in \mathbb{R}^n$. We now present the learning algorithm, then the policy parameterization.

**On-Line POMDP** The On-Line POMDP policy-gradient algorithm (OLPOMDP) (Baxter, Bartlett, and Weaver 2001), and many similar algorithms (Williams 1992; Szepesvári 2009), maximize the long-term average reward

$$R(\theta) := \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_\theta \left[ \sum_{t=1}^{T} r(s_t) \right], \qquad (1)$$

where the expectation $\mathbb{E}_\theta$ is over the distribution of state trajectories $\{s_0, s_1, \ldots\}$ induced by the transition matrix and the policy. To maximize $R(\theta)$, *goal* states must be reached as frequently as possible. This simultaneously minimizes plan duration and maximizes the probability of reaching the goal (failure states achieve no reward).

A typical gradient ascent algorithm would repeatedly compute the gradient $\nabla_\theta R$ and follow its direction. Because an exact computation of the gradient is very expensive in our setting, OLPOMDP relies on Monte-Carlo estimates generated by simulating the task. At each time step of the simulation loop, it computes a one-step gradient $\mathbf{g}_t = r_t \mathbf{e}_t$ and immediately updates the parameters in the direction of $\mathbf{g}_t$. The eligibility vector $\mathbf{e}_t$ contains the discounted sum of normalized action probability gradients. At each step, $r_t$ indicates whether to move the parameters in the direction of $\mathbf{e}_t$ to promote recent actions, or away from $\mathbf{e}_t$ to deter recent actions (Algorithm 1).

**Algorithm 1**: OLPOMDP FPG Gradient Estimator

Set $s_0$ to initial state, $t = 0, \mathbf{e}_t = [0]$, init $\theta_0$ randomly
**while** $R$ *not converged* **do**
  Compute distribution $\mathbb{P}[a_t = i|\mathbf{s}_t; \theta_t]$
  Sample action $i$ with probability $\mathbb{P}[a_t = i|\mathbf{s}_t; \theta_t]$
  $\mathbf{e}_t \leftarrow \beta \mathbf{e}_{t-1} + \nabla \log \mathbb{P}[a_t|\mathbf{s}_t; \theta_t]$
  $s_{t+1} \leftarrow \text{next}(s_t, i)$
  $\theta_{t+1} \leftarrow \theta_t + \alpha r_t \mathbf{e}_t$
  **if** *isTerminalState*$(s_{t+1})$ **then** $s_{t+1} \leftarrow s_0$
  $t \leftarrow t + 1$

OLPOMDP is "on-line" because it updates parameters for every non-zero reward. It is also "on-policy" in the RL sense of requiring trajectories to be generated according to $\mathbb{P}[\cdot|\mathbf{s}_t; \theta_t]$. Convergence to a (possibly poor) locally optimal policy is guaranteed even if some state information is omitted from $\mathbf{s}_t$ for simplifying the policy representation.

**Linear-Network Factored Policy** FPG's policy is *factored*: it is made of one linear network per action, each of them taking the same vector $\mathbf{s}$ as input (plus a constant 1 bit to provide bias to the perceptron) and outputting a real value $f_i(\mathbf{s}_t; \theta_i)$. In a given state, a probability distribution over eligible actions is computed as a Gibbs[1] distribution

$$\mathbb{P}[a_t = i|\mathbf{s}_t; \theta] = \frac{exp(f_i(\mathbf{s}_t; \theta_i))}{\sum_{j \in \mathcal{A}} exp(f_j(\mathbf{s}_t; \theta_j))}.$$

The interaction loop connecting the policy and the task is represented in Figure 1. Initially, the parameters are set to 0, giving a uniform random policy; encouraging exploration of the action space. Each gradient step typically moves the parameters closer to a deterministic policy.

Due to the availability of benchmarks and compatibility with LAMA we focus on the non-temporal IPC version of FPG. The temporal version extension simply gives each action a separate Gibbs distribution to determine if it will be executed, independently of other actions (mutexes are resolved by the simulator).

[1] Essentially the same as a Boltzmann or soft-max distribution.

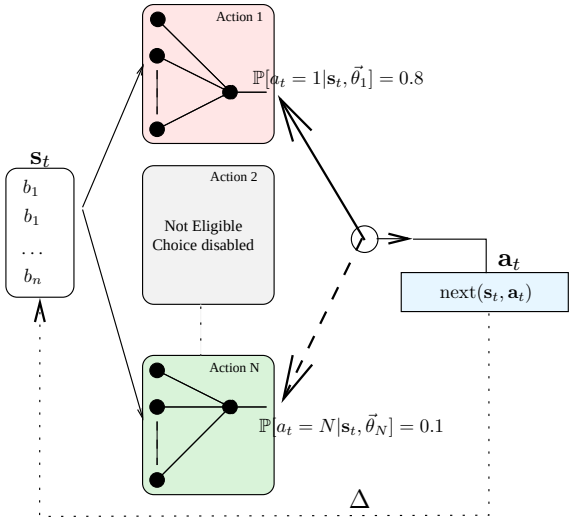


Figure 1: FPG seen as a factored controller made of multiple linear networks

## Reward Shaping

In Reinforcement Learning, shaping is the idea of modifying the sequential decision problem at hand to make it simpler. Reward shaping (rewritting the reward function, i.e. the objective) (Dorigo and Colombetti 1994; Matarić 1994) is the main research direction in this area, but it is also possible to modify the physics of the system (through the transition function) (Randløv 2000). Major difficulties when using shaping approaches are:

- to possibly automate the shaping process;
- to make sure that the modified problem is indeed simpler to solve than the original one;
- to make sure that (1) either one really benefits from shaping when progressively turning the modified problem into the original one (e.g., this is not the case in the mountain-car problem when starting with a flat mountain); (2) or an optimal solution to the modified problem is also an optimal solution of the original problem (which is possible for certain shaped rewards (Ng, Harada, and Russell 1999)).

We focus in this paper on reward shaping – which is already used in FPG – to theoretically verify its usage conditions and experiment with various automatically computed (heuristic-based) reward shapers.

### Known Results

**Simple Results for Finite or Infinite Horizon Problems** A very common result is that, in a classical MDP (i.e., with an infinite horizon and a discounted reward), policy invariance is guaranteed under any linear transformation $r' = pr + q$, where $p > 0$ and $q \in \Re$ ($r'$ is the *shaped* reward and $\rho$ the *shaping* reward). This property still holds for a finite horizon problem or with an average reward criterion. In the case of an indefinite horizon problem (e.g., in probabilistic planning), the optimal policy may change, being attracted by (respectively avoiding) looping behaviors if $q$ has a large positive (resp. negative) value. This leaves us with the possibility of tuning $p$, which (1) has no influence on the speed of convergence of a value function estimator, and (2) is equivalent in a policy gradient algorithm to tuning the learning step size ($\alpha$ in FPG).

**Potential-based Shaping** A more useful result (Ng, Harada, and Russell 1999) for classical MDPs is that, given a potential function $\Phi(s)$ that maps states into numbers, and $\rho(s, a, s') = \gamma\Phi(s') - \Phi(s)$, the transform $r'(s, a, s') = r(s, a, s') + \rho(s, a, s')$ is policy invariant. This extends to undiscounted MDPs with one *absorbing state* $s_{\text{abs}}$ (finite or indefinite horizon problems with a total reward criterion, i.e., $\gamma = 1$) and if all policies are *proper*, i.e., if $s_{\text{abs}}$ is reached with probability 1.

The reader accustomed to heuristic search will have thought "heuristic" when reading "potential function" above. Indeed it is not difficult to see that the two approaches – reward shaping by $\Phi$ vs. initialization of $Q$-values by $\Phi$ – are similar for $Q$-value based algorithms, and even equivalent for $Q$-learning or sarsa (Wiewiora 2003). Note however that this equivalence is limited since it does not apply to algorithms like FPG that are not based on learning of $Q$-values. This notwithstanding, we can often exploit the correspondence potential/heuristic function in such algorithms as well. We do so herein for FPG.

**Reward Definition for Probabilistic Planning** In probabilistic planning, i.e., when trying to maximize the probability to reach a goal state – here, without minimizing any cost –, a typical reward function will distinguish three reward values depending on whether the next state is (i) a success state ($r_{\text{succ}}$), (ii) a failure state ($r_{\text{fail}}$), or (iii) any other state ($r_o$). With this scheme, as there is typically no guarantee on the plan length, the default reward should be $r_o = 0$. Then, since a trajectory may end with success or failure, or may never end, the expected cumulative reward for a state $s$ is:

$$V(s) = r_{\text{succ}}.Pr(\text{succeed}|s) + r_{\text{fail}}.Pr(\text{fail}|s) + 0.Pr(\text{never end}|s). \quad (2)$$

In problems where an absorbing (/terminal) state is always attained (e.g., because of limited resources or time), we have, for all state $s$, $Pr(\text{never end}|s) = 0$ and $Pr(\text{succeed}|s) + Pr(\text{fail}|s) = 1$, so that Equation 2 can be rewritten: $V(s) = [r_{\text{succ}} - r_{\text{fail}}].Pr(\text{succeed}|s) + r_{\text{fail}}$, implying the constraint $[r_{\text{succ}} - r_{\text{fail}}] > 0$. Otherwise, whenever infinite executions are possible, a non zero failure reward $r_{\text{fail}}$ is either useless or detrimental, so that a sensible rewarding scheme is $r_{\text{fail}} = 0$ and $r_{\text{succ}} > 0$, the criterion at hand being $V(s) = r_{\text{succ}}.Pr(\text{succeed}|s)$.

### Application to our Setting

To apply Ng et al.'s potential-based shaping in our probabilistic planning setting, a first requirement is that our problem have a finite or indefinite horizon, which is ensured by defining a maximum plan length. Then, an issue is that we have multiple absorbing states. This is solved by observing that this situation is equivalent to rewriting the planning problem in such a way that success and failure states are not

absorbing, but all lead to a single absorbing state. In practice, it is more convenient to simply ensure that all success and failure states are mapped to the same potential value: $\forall s \in (S_{\text{succ}} \bigcup S_{\text{fail}}),\ \Phi(s) = \Phi_{\text{abs}}$.

Then, a first remark regarding the heuristic (/potential) function is that it does not need to be admissible for the shaping to be policy invariant. Admissibility may be a requirement depending on the resolution algorithm, though. A more important issue is that some heuristic functions known in planning are actually *pseudo-heuristic* functions whose value depends not only on the current state, but also on past states and actions. Indeed, landmark heuristics are computed based on events that have occurred since the beginning of the current trajectory. As a result, the corresponding shaped reward $r'$ is non-Markovian (Thiébaux et al. 2006).[2] Two interesting open questions are whether this reward shaping is policy invariant and, if so, which MDP solution algorithms still work. An encouraging property suggesting that policy invariance holds is that, in this particular case, the cumulative shaping reward over a complete trajectory only depends on the start and end states: $\rho(s_0, a_0, s_1) + \cdots + \rho(s_K, a_K, s_{\text{abs}}) = r(s_{abs}) + \Phi(s_{abs}) - \Phi(s_0)$, where $r(s_{abs}) = r_{succ}$ or $r_{fail}$.

**Application to FPG** In our setting, all executions terminate in finite time, so that the reward for the terminal states only have to verify: $r_{succ} - r_{fail} > 0$. At the end of a trajectory, the accumulated shaping reward is always $[\Phi(s_{abs}) - \Phi(s_0)]$, always indicating a progress towards the goal. We have decided to compensate for this phenomenon (i.e., to ensure that a failing trajectory gets zero reward) by setting $r_{fail} = -[\Phi(s_{abs}) - \Phi(s_0)]$.

A second point is that OLPOMDP does not optimize a cumulative reward, but a (biased) long-term average reward. To properly maximize a success probability with FPG, a solution is then to (i) set $\beta = 1$ and (ii) reset the eligibility vector after reaching an absorbing state. However, in our implementation we stick to the default biased approach as experiments show that it is more efficient in practice (it facilitates credit assignment).

## Landmarks Heuristic Functions

Starting with the basic notion of landmarks and the mechanism we use to find them, we herein explain the original LAMA heuristic function as well as two variants we designed, improving on various aspects of that function.

### Landmarks and Orderings

Given a probabilistic planning task $\mathcal{T}$, a *landmark* is a literal that must be true at some point on every successful trajectory, i.e., on every path leading from the initial state $s_0$ to a state that satisfies the goal $G$. This definition is the straightforward generalization of the original proposal made in classical planning (Hoffmann, Porteous, and Sebastia 2004). The heuristic functions we design herein are, indeed, entirely based on classical planning, i.e., we re-use and extend existing classical planning techniques to obtain our reward shaper to be used in FPG. Precisely, we start from the techniques implemented in LAMA, one of the winners of the 2008 planning competition, whose main novel contribution is a heuristic function $h^{\mathsf{LM}}$ based on landmarks (Richter, Helmert, and Westphal 2008).

LAMA's pre-processor automatically identifies a set $\mathcal{L}$ of landmarks as well as a set $\mathcal{O}$ of ordering constraints between them.[3] Each of the latter constraints has one of three possible types: $l_1 <_n l_2$ ("natural"), $l_1 <_{gn} l_2$ ("greedy necessary"), $l_1 <_r l_2$ ("reasonable"). We do not have the space to explain in detail how these are derived. LAMA guarantees the following properties:

(A) $l_1 <_n l_2 \implies l_2$ is not true initially, and if a plan adds $l_2$ at time $t \geq 1$, then $l_1$ must be true at some time $t' < t$. For example, in Logistics, to have a package at the airport of its destination city, we must previously have had it at the airport of its (different) origin city.

(B) $l_1 <_{gn} l_2 \implies l_2$ is not true initially, and if a plan first adds $l_2$ at time $t \geq 1$, then $l_1$ must be true at time $t - 1$. For example, in Blocksworld, to stack block $b$ onto block $b'$ we must be holding $b$.

(C) $l_1 <_r l_2 \implies$ if a plan achieves $l_2$ strictly before $l_1$, then achieving $l_1$ will involve deleting $l_2$, and re-achieving $l_2$ later. For example, in Blocksworld, "on(B,C)" $<_r$"on(A,B)".

In addition, LAMA guarantees (D) that the transitive hull of the orderings is irreflexive, i.e., there are no cycles.

Note the methodological difference between $l_1 <_n l_2$ and $l_1 <_{gn} l_2$ orders vs. $l_1 <_r l_2$ orders. (A) and (B) guarantee that natural respectively greedy necessary orders are *strictly sound*, i.e., every solution plan is guaranteed to make $l_1$ true strictly before it makes $l_2$ true. By contrast, reasonable orders $l_1 <_r l_2$ are *optional* in the sense that (C) does not hinder the plan from violating the order (and paying the cost of achieving $l_2$ twice). As we will see, the distinction between sound and optional orders is of relevance for certain properties of LAMA's heuristic function; it is especially important for admissible heuristics (Karpas and Domshlak 2009) which cannot take into account optional orders.[4]

We run LAMA on the "all-outcomes determinization" (Yoon, Fern, and Givan 2007), $\mathcal{T}^D$, of the probabilistic planning task $\mathcal{T}$. $\mathcal{T}^D$ inserts one deterministic action for every possible outcome of every action in $\mathcal{T}$. It is obvious that any landmark in $\mathcal{T}^D$ is also a landmark in $\mathcal{T}$, and that any

[2] The problem is an NMRDP and can therefore be turned back into an MDP by extending the state with information about past events. From this point of view, the original state is just a partial observation of the extended state, and it is a good thing that FPG relies on an algorithm that is valid for POMDPs.

[3] LAMA allows "disjunctive" landmarks, where one of a set of literals must be true in any plan. Our techniques also handle this. For simplicity, we consider only single-literal landmarks here.

[4] Importantly, (A), (B) require $l_2$ to be false in the initial state. $l_1 <_n l_2$ and $l_1 <_{gn} l_2$ orders where $l_2$ is initially true can be meaningful for heuristics if $l_2$ may be deleted at some point (e.g. $l_2$=on$(b, b')$ and $l_1$=holding$(b)$); but clearly they cannot be strictly sound. One can even construct examples where $l_1 <_{gn} l_2$ and, on every plan, $l_2$ can only be true in states *strictly preceding* $l_1$, so every plan violates the order in a strong sense. It remains future work to see if such orders may be useful for satisficing planning.

ordering $l_1 <_X l_2$, $X \in \{n, gn, r\}$ that is valid in $\mathcal{T}$ according to (A)–(C) as applies, is valid in the same sense in $\mathcal{T}$ as well (where "plan" in (A)–(C) translates into "successful trajectory"). The reason for this is simply that (A)–(C) are statements on the set of successful trajectories, and every such trajectory for $\mathcal{T}$ corresponds to a successful trajectory for $\mathcal{T}^D$. The question then is how we can turn this information into a heuristic function.

## LAMA Heuristic: $h^{\mathsf{LM}}$

LAMA's heuristic is $h^{\mathsf{LM}}(s,\pi) := |\mathcal{L} \setminus (a^{\mathsf{LM}}(s,\pi) \setminus r^{\mathsf{LM}}(s,\pi))|$ counting the landmarks that are either not "accepted" ($a^{\mathsf{LM}}(s,\pi)$) or that are accepted but "required again" ($r^{\mathsf{LM}}(s,\pi)$). $s$ is the evaluated state and $\pi$ is the sequence of states traversed by the plan. LAMA defines $a^{\mathsf{LM}}(s,\pi) :=$

$$\begin{cases} \{l \in \mathcal{L} \mid s \models l, l^{\leftarrow} = \emptyset\} & \pi = \langle s_0 \rangle \\ a^{\mathsf{LM}}(p,\pi) \cup & \pi = \langle \dots, p, s, \dots \rangle \\ \{l \in \mathcal{L} \mid s \models l, \forall l' \in l^{\leftarrow} : l' \in a^{\mathsf{LM}}(p,\pi)\} & \end{cases} \quad (3)$$

where $l^{\leftarrow}$ is the set of landmarks $l'$ ordered before $l$. A landmark is accepted if it is true, and all its predecessors were accepted beforehand already; an accepted landmark remains accepted. The set $r^{\mathsf{LM}}(s,\pi)$ is the union of **false-goal**:

$$\{l \in \mathcal{L} \mid s \not\models l, G \models l\} \quad (4)$$

and of **open-prerequisite**:

$$\{l \in \mathcal{L} \mid s \not\models l, \exists l' \in l^{\rightarrow_{gn}} : l' \notin a^{\mathsf{LM}}(s,\pi)\} \quad (5)$$

where $l^{\rightarrow_{gn}}$ denotes the set of landmarks $l'$ such that $(l <_{gn} l') \in \mathcal{O}$. Both rules capture landmarks $l$ that are currently false and that have a reason to become true again. In (4), the reason is simply because $l$ is a top-level goal. In (5), the reason is another landmark $l'$ that is not yet accepted and that requires $l$ to be true immediately beforehand.

While these definitions seem sensible at first sight, we will see now that they have a number of shortcomings; we will also see how to address those.

## Improved LAMA Heuristic: $h^{\mathsf{iLM}}$

Consider the mechanics by which landmarks are accepted. This basically says that the plan must comply with the orders $\mathcal{O}$. But what does it mean to "comply with" $\mathcal{O}$? LAMA assumes that "$l_1$ is true strictly before $l_2$" for each order $l_1 < l_2$. That does correspond to the meaning (A), (B) of $<_n$ and $<_{gn}$ orders. But it does not correspond to the meaning (C) of reasonable orders $l_1 <_r l_2$: (C) allows us to achieve $l_1$ and $l_2$ simultaneously, and $l_2$ may be true in the initial state; $l_1 <_r l_2$ makes sense in both cases.[5] Hence a trajectory $\pi$ should be considered *compliant* if, for every $l_1 <_r l_2 \in \mathcal{O}$, $l_1$ is true at least as early as $l_2$. Then:

**Proposition 1** *Assume guarantees (A)–(D). There exist a task $\mathcal{T}$ and a compliant trajectory $\pi$ ending in state $s$ so that $s \models G$ and $h^{\mathsf{LM}}(s,\pi) > 0$.*

[5] We remark that (C) also allows pathological cases, like $l <_r l$. LAMA does not return such orders.

A simple example proving Proposition 1 is a Blocksworld task where a size-$k$ goal stack is already assembled in the initial state. This task has $k-1$ goals of the form "on($b_i, b_{i+1}$)", with a reasonable order between each consecutive pair. The definition of $a^{\mathsf{LM}}(s,\pi)$ forces us to leave this stack untouched for $k-1$ steps before all goals are accepted. Similar examples can be constructed based on actions that achieve literals with a $<_r$ order within a single effect. Both can happen in LAMA.

Note that the observed phenomenon affects the quality of $h^{\mathsf{LM}}$ far beyond "only" goal states. It may cause estimation errors throughout the search space, starting at the initial state. We fix this in an improved heuristic $h^{\mathsf{iLM}}(s) := |\mathcal{L} \setminus (a^{\mathsf{iLM}}(s,p) \setminus r^{\mathsf{iLM}}(s,\pi))|$ where $a^{\mathsf{iLM}}(s,\pi) :=$

$$\begin{cases} \{l \in \mathcal{L} \mid s \models l, & \pi = \langle s_0 \rangle \\ \quad \forall l' \in l^{\leftarrow} : l' \in a^{\mathsf{iLM}}(s,\pi)\} & \\ a^{\mathsf{LM}}(p,\pi) \cup & \pi = \langle \dots, p, s, \dots \rangle \\ \{l \in \mathcal{L} \mid s \models l, \forall l' \in l^{\leftarrow} : l' \in a^{\mathsf{LM}}(p,\pi) \cup a^{\mathsf{LM}}(s,\pi)\} & \end{cases} \quad (6)$$

This definition recurses, in each case, onto itself to allow newly accepted landmarks to be ordered with respect to each other. This recursion terminates due to guarantee (D). Assuming for the moment that $r^{\mathsf{iLM}}(s,\pi) = r^{\mathsf{LM}}(s,\pi)$ (this will be modified further below), we get:

**Proposition 2** *Assume guarantees (A)–(D). For any task $\mathcal{T}$ and compliant trajectory $\pi$ ending in state $s$ so that $s \models G$, we have $h^{\mathsf{iLM}}(s,\pi) = 0$.*

This is because, in $s$, all landmarks will be accepted.[6]

Since $<_r$ orders are optional, Proposition 2 is the strongest result of this form that we can get, for this kind of heuristic. The opposite of Proposition 2, i.e. that the heuristic value is non-zero for non-goal states, trivially holds for both $h^{\mathsf{LM}}$ and $h^{\mathsf{iLM}}$ due to the **false-goal** rule.

We next note that $h^{\mathsf{LM}}$ is also overly restrictive about marking landmarks as required again. First, consider this new rule **doomed-goal**:

$$\{l \in \mathcal{L} \mid s \models l, G \models l, \exists l' \in l^{\rightarrow_{gn!}} : l' \notin a^{\mathsf{LM}}(s,\pi)\} \quad (7)$$

where $l^{\rightarrow_{gn!}}$ denotes the set of landmarks $l'$ such that $(l <_{gn} l') \in \mathcal{O}$ and $l'$ is inconsistent with $l$. In this case, $l$ is a currently satisfied goal, but one of its successors has not yet been achieved, and in the process of doing so $l$ will be deleted. Clearly, we can add **doomed-goal** into the definition of $r^{\mathsf{iLM}}(s,\pi)$ without invalidating Proposition 2. We remark that, while the **doomed-goal** rule may appear a bit artificial, it is actually omnipresent at least in the Blocksworld. In some probabilistic versions used in the competition, "hand-empty" is a goal. The heuristic then oscillates by 1 between every two states because LAMA considers "hand-empty" to be dealt with whenever it is true. This behavior stops when using **doomed-goal**, which correctly determines that having achieved "hand-empty" is useless if we need to consume it for arranging additional blocks.

We finally use the rule **required-ancestor**:

[6] An alternative way of guaranteeing this is to disallow (and not generate) $l_1 <_r l_2$ if $l_1$ and $l_2$ can become true at the same time. However, such orders can be useful for heuristic estimation. We get back to this when discussing future work in the conclusion.

$$\{l \in \mathcal{L} \mid s \not\models l, \exists l' \in l^{\rightarrow_{gn}} : l' \in r^{\mathsf{iLM}}(s,\pi)\} \cup \{l \in \mathcal{L} \mid s \models l, G \models l, \exists l' \in l^{\rightarrow_{gn!}} : l' \in r^{\mathsf{iLM}}(s,\pi)\} \quad (8)$$

This rule induces a kind of transitive closure, stating that, if landmark $l'$ is required again, then so are the predecessors needed to establish $l'$ (again we rely on (D) for the recursion to terminate). Clearly, this does not invalidate Proposition 2 since, to fire, i.e. to include any new literals, this rule requires one of the other rules to fire in the first place. We remark that the rule slightly abuses the meaning of $<_{gn}$ orders, by using them as if $l_1$ was *always* needed to establish $l_2$ – a constraint termed "necessary order" (Hoffmann, Porteous, and Sebastia 2004) – whereas by (B) they are guaranteed to be needed only when $l_2$ is established *for the first time*. This abuse appears to be rather harmless, especially in the light that necessary orders are frequent in planning benchmarks, and $<_{gn}$ orders (like, the need to hold a block before stacking it onto another one) are indeed often necessary.

### Markovian LAMA Heuristic: $h^{\mathsf{mLM}}$

All heuristics proposed thus far in the literature based on detecting landmarks and orders in a pre-process (Richter, Helmert, and Westphal 2008; Karpas and Domshlak 2009) are pseudo-heuristics, or "non-Markovian" in the sense that they depend on the history of a state, not only on the state itself.[7] One might get the impression that such heuristics are necessarily not Markovian. However, that is actually not the case. Given our previous observations regarding policy invariance, this is of potential importance.

We experiment with a simple Markovian variant $h^{\mathsf{mLM}}$, defined by $h^{\mathsf{LM}}(s) := |r^{\mathsf{mLM}}(s)|$ where $r^{\mathsf{mLM}}(s) :=$

$$\{l \in \mathcal{L} \mid s \not\models l, G \models l\} \cup \{l \in \mathcal{L} \mid s \not\models l, \exists l' \in l^{\rightarrow_{gn}} : l' \in r^{\mathsf{mLM}}(s)\} \cup \{l \in \mathcal{L} \mid s \models l, G \models l, \exists l' \in l^{\rightarrow_{gn!}} : l' \in r^{\mathsf{mLM}}(s)\} \quad (9)$$

This heuristic starts at the unsatisfied goals and then uses the $<_{gn}$ orders to identify additional literals that will still have to be made true. Note that this inherits from **required-ancestor** the "abuse" of $<_{gn}$ orders as necessary orders. Note also that the $<_n$ orders returned by LAMA are *not* suitable for use in $r^{\mathsf{mLM}}(s)$ because they refer to long-term (more than 1 step) dependencies. Consider the situation where $l_1 <_{gn} l_2$, $l_2 <_{gn} l_3$, $l_1 <_n l_3$, and achieving $l_i$ involves deleting $l_{i-1}$. If we are in a state where $l_3 \in r^{\mathsf{mLM}}(s)$ and $s \models l_2$, then using the order $l_1 <_n l_3$ will overlook the intermediate literal $l_2$ and will wrongly conclude that we still need to achieve $l_1$.

**Proposition 3** *Assume guarantees (A)–(D). For any task $\mathcal{T}$ and state $s$, we have that $s \models G$ if and only if $h^{\mathsf{mLM}}(s) = 0$.*

So, apart from the fact that the heuristic now depends only on the current state, we get a stronger validity property regarding goal states, that does not force us to comply with reasonable orders. The downside is, of course, that we bought this validity improvement at the cost of not actually making any use of reasonable orders.

[7] The LM-cut heuristic (Helmert and Domshlak 2009) is also derived based on a form of landmarks, but a very different one that detects disjunctive action landmarks anew for every evaluated state.

## Experiments

We implemented the new landmarks heuristics $h^{\mathsf{iLM}}$ and $h^{\mathsf{mLM}}$ in C++, within the non-temporal IPC version of FPG.[8] We also implemented the original LAMA heuristic function $h^{\mathsf{LM}}$. All these functions rely on creating the all-outcomes determinization of the PPDDL task at hand, and calling LAMA in order to compute the landmarks and orderings. These are then parsed into FPG, and the heuristics are computed internally (i.e. not by calling LAMA). We also implemented a pipe-based interface to FF that allows to retrieve the relaxed plan heuristic $h^{\mathsf{FF}}$ for any given state (without having to re-start FF for every single state which would of course be wasteful). Finally, FPG already contained the option to shape rewards by counting goals, denoted $h^{\mathsf{G}}$.

The experiments where run on a Core2Quad CPU at 2.4 GHz, using a single core, with a learning time of 10 minutes (memory is not an issue for FPG; we allowed 1GB). We measure performance in terms of the success rate of the policy computed by FPG within these resources, with the mentioned variants of the reward shaping and with no shaping at all. We use a subset of the IPPC benchmarks 2004–2008. The rationale for choosing the subset was to use only those domains where LAMA found a significant number of landmarks and orderings; some domains we could not consider due to difficulties with LAMA's parser. We ended up using Blocksworld (IPPC 2006), Exploding-Blocksworld (IPPC 2006), Schedule (IPPC 2006), Zenotravel (IPPC 2006), and Triangle-Tireworld (IPPC 2008). We also ran experiments on the classical Gripper domain (IPC 1998) – this is interesting in that it provides a test bed for which we know the heuristics are highly informative.

FPG has a number of search parameters that can have – and did have in our experiments – a large impact on performance: the learning rate $\alpha$, the eligibility discount factor $\beta$, the goal reward $r_{\text{succ}}$, and a constant $H$ by which the rewards obtained from the heuristic function are scaled. We experimented with 12 configurations of these parameters in total, comprising 3 different settings of $\beta$ (0.85, 0.9, 0.95), 3 settings of $r_{\text{succ}}$ (1000, 100, 10), 3 settings of $H$ (100, 10, 1), and 4 settings of $\alpha$ (0.0001 as well as 3 versions where $\alpha$ becomes larger respectively smaller over time). While performance varies a lot over these configurations, the results are largely inconclusive (understanding the impact of the parameters would entail much broader experimentation), and their analysis would not serve our primary goal of evaluating the different heuristic functions. We hence show only summary results in terms of the minimum, average, and maximum success rate across the 12 parameter settings.

Before looking at the success rates, let us first provide some data confirming our initial intuitions why landmarks counting may be a good idea (why this "glitters", in the terms of the title). Consider first Table 2, which gives the

[8] When finishing up this paper, we noticed that our implementation of $h^{\mathsf{iLM}}$ differs from its definition herein in that it's missing the recursion onto $a^{\mathsf{LM}}(s,\pi)$ in the bottom case of Equation 6. Thus Proposition 2 does not hold for the implemented version in case there are actions whose effect contains two landmarks $l_1, l_2$ with $l_1 <_r l_2$. In the benchmarks used here, such actions do not occur.

| (#atoms, #actions) | $h^{\mathsf{G}}$ | $h^{\mathsf{LM}}$ | $h^{\mathsf{iLM}}$ | $h^{\mathsf{mLM}}$ | $h^{\mathsf{FF}}$ | FF |
|---|---|---|---|---|---|---|
| bw 08 (89, 1224) | 8 | 19 | 20 | 20 | 16 | 20 |
| bw 12 (181, 3900) | 10 | 22 | 23 | 23 | 19 | 48 |
| ebw 05 (52, 90) | 5 | 13 | 13 | 13 | 10 | 16 |
| ebw 08 (106, 216) | 6 | 15 | 15 | 15 | 12 | 16 |
| ebw 10 (152, 330) | 7 | 23 | 25 | 25 | 16 | 30 |
| grip 12 (68, 100) | 12 | 25 | 25 | 25 | 25 | 35 |
| grip 16 (88, 132) | 16 | 33 | 33 | 33 | 33 | 47 |
| sched 07 (102, 83) | 0 | 7 | 7 | 0 | 7 | 16 |
| sched 09 (275, 242) | 0 | 9 | 9 | 0 | 9 | 24 |
| sched 11 (198, 227) | 0 | 11 | 11 | 0 | 11 | – |
| t-tire 02 (49, 33) | 1 | 4 | 4 | 4 | 4 | 4 |
| t-tire 04 (153, 107) | 1 | 4 | 4 | 4 | 8 | 8 |
| t-tire 06 (313, 221) | 1 | 4 | 4 | 4 | 12 | 12 |
| zeno 03 (98, 1208) | 2 | 12 | 12 | 12 | 16 | 18 |
| zeno 05 (114, 1788) | 2 | 11 | 11 | 11 | 14 | 16 |

Table 2: Initial distance to the goal.

initial state distance estimate computed by each heuristic, as well as the FF plan length as a kind of "ideal quality measure".[9] We see that, as expected, in all examples tested the heuristic values obtained by counting landmarks are much more informed than those obtained by counting goals. The landmarks heuristics are most often equally strong, with a slight advantage for our two new versions in the two Blocksworld-based domains. Interestingly, in the Blocksworld domains, the landmarks heuristics are much stronger than $h^{\mathsf{FF}}$; in the other domains, as one would expect, the much more costly $h^{\mathsf{FF}}$ heuristic usually dominates.[10]

| | – | $h^{\mathsf{G}}$ | $h^{\mathsf{LM}}$ | $h^{\mathsf{iLM}}$ | $h^{\mathsf{mLM}}$ | $h^{\mathsf{FF}}$ |
|---|---|---|---|---|---|---|
| bw 08 | 7916 | 6303 | 6442 | 6115 | 4702 | 2419 |
| bw 12 | 2454 | 1314 | 1339 | 1880 | 1722 | 828 |
| ebw 05 | 61791 | 66941 | 14966 | 23316 | 13858 | 4950 |
| ebw 08 | 30733 | 31491 | 5881 | 4025 | 6283 | 2826 |
| ebw 10 | 23216 | 24041 | 6783 | 13500 | 6100 | 2593 |
| grip 12 | 36438 | 28545 | 21538 | 21411 | 21227 | 4771 |
| grip 16 | 35003 | 20805 | 15772 | 15587 | 15473 | 3953 |
| sched 07 | 31941 | 33324 | 26908 | 26083 | 29731 | 3158 |
| sched 09 | 7666 | 8312 | 6208 | 7216 | 7647 | 1395 |
| sched 11 | 9766 | 10464 | 8825 | 8425 | 9165 | 1545 |
| t-tire 02 | 81066 | 80850 | 63798 | 60073 | 59522 | 6303 |
| t-tire 04 | 28325 | 27533 | 20405 | 20774 | 21523 | 3428 |
| t-tire 06 | 13766 | 13783 | 8878 | 9074 | 9438 | 1923 |
| zeno 03 | 8718 | 8552 | 6495 | 5080 | 5040 | 2695 |
| zeno 05 | 6105 | 6214 | 4983 | 3837 | 4036 | 2084 |

Table 3: Average number of simulation steps per second.

Do the more informed heuristics cost us a lot more computation time? Table 3 provides the answer, in terms of the rate at which FPG generates nodes, i.e., the average number of simulation steps per second. The result is largely a confirmation of our initial intuitions – counting landmarks does not induce a big computational overhad. In most cases, the generation rate when counting landmarks is at about 70% of that when counting goals. The only notable exception to this are the Blocksworlds (landmarks cause basically no overhead in the basic version, but reduce the generation rate to around 20% in Exploding-Blocksworld). Looking at the behavior of $h^{\mathsf{FF}}$, the speed comparison is impressively in favor of landmarks, by factors of at least 2 and up to almost 10. This is especially significant in the light of Table 2, which shows that – as far as initial states are concerned – the quality gap between these heuristics is not so large.

Having pointed out all these fairly good news (for landmarks heuristics), the time has come for us to bring up the bad news. Table 4 shows the success rates of FPG, with all forms of reward shaping tested. In each table entry, we show the minimum/average/maximum success rate, i.e., the percentage of times (out of 100 test executions) that executing the policy lead to a goal state within 10000 steps. For readability, the average rates are shown in boldface.

The most striking observation in Table 4 is – that there is not much to observe! For a start, look only at the average values, for the trivial $h^{\mathsf{G}}$ compared to the quite non-trivial landmarks and $h^{\mathsf{FF}}$ heuristics. In classical planning, the performance difference between these heuristics is *huge*. Not so here: the maximum difference that arises anywhere is 38% in Zenotravel 05 between $h^{\mathsf{G}}$ and $h^{\mathsf{LM}}$ – to the advantage of $h^{\mathsf{G}}$. So the non-trivial heuristics don't buy us much. Worse, the *only* cases where any of them fares convincingly better than trivial goal counting are Blocksworld and Gripper. Since Gripper is not actually a probabilistic domain, this reduces to a single "good case". The final straw, as far as landmarks are concerned, is that this good case pertains only to $h^{\mathsf{FF}}$. (Unless one counts the 77% average of $h^{\mathsf{iLM}}$ vs. the 65% average of $h^{\mathsf{G}}$ in Blocksworld 08 as a significant improvement; but we're not quite that desparate yet.)

There is no "gold" to be found here. Why? Answering this question with confidence would require much more detailed experiments testing particular hypotheses. What we can offer at this point are some speculative intuitions.

The results are easiest to understand for $h^{\mathsf{FF}}$ in Triangle-Tireworld. The relaxed plan always chooses the shortest path to the goal location, enticing the learner to prefer the same route. However, the domain is designed in a way making this route (particularly in the initial state) very dangerous. This leads to the dramatically bad performance of $h^{\mathsf{FF}}$. The landmarks heuristics do better here not because they are more clever about the danger but because they are less good at capturing the structure of the determinized planning problem and hence provide a weaker guidance to the learner. This results in more random walk, and thus in a better chance to find a safe route. A contrasting behavior can be observed in Gripper, where $h^{\mathsf{FF}}$ is very good and hence is a comparatively very reliable guidance. It is a mystery to us why none of the heuristics yields an advantage in Zenotravel, where the heuristics are fairly good quality. As for Schedule, judging from the inability of FF to solve the largest instance, it seems that the heuristics are very bad even in the determinized version of this domain, so their failure to provide good guidance on the probabilistic version makes

[9] The real "ideal measure" would be the expected number of steps to the goal. This is not known for many of the tested benchmarks. Classical plan length is a viable alternative since the heuristics are classical too. That said, it would be better to compare to optimal plan length, where known; we will do in future work.

[10] There is no value for FF in Schedule 11 because, there, FF actually runs out of time while trying to solve the determinized planning task – which is quite interesting in itself because this task is fairly small and, as we will see, the goal in this task can be feasibly reached by random walks, so this task is an interesting challenge to heuristics for classical planning.

| | – | $h^{\mathsf{G}}$ | $h^{\mathsf{LM}}$ | $h^{\mathsf{iLM}}$ | $h^{\mathsf{mLM}}$ | $h^{\mathsf{FF}}$ |
|---|---|---|---|---|---|---|
| bw 08 | 0/ **9**/100 | 0/**65**/100 | 2/**70**/100 | 4/**77**/100 | 3/**72**/100 | 1/**83**/100 |
| bw 12 | 0/ **0**/ 0 | 0/ **0**/ 4 | 0/ **0**/ 1 | 0/ **0**/ 0 | 0/ **0**/ 0 | 0/**24**/ 92 |
| ebw 05 | 0/**48**/ 64 | 0/**29**/ 57 | 0/**28**/ 60 | 0/ **0**/ 0 | 0/ **6**/ 48 | 0/**26**/ 56 |
| ebw 08 | 0/ **0**/ 0 | 20/**55**/ 69 | 46/**55**/ 66 | 0/**36**/ 65 | 0/**22**/ 70 | 55/**62**/ 67 |
| ebw 10 | 0/ **0**/ 0 | 0/ **0**/ 0 | 0/ **0**/ 0 | 0/ **0**/ 0 | 0/ **0**/ 0 | 0/ **0**/ 0 |
| grip 12 | 15/**91**/100 | 51/**91**/100 | 43/**95**/100 | 37/**92**/100 | 52/**89**/100 | 46/**92**/100 |
| grip 16 | 0/**64**/100 | 0/**68**/100 | 9/**73**/100 | 10/**67**/100 | 0/**61**/100 | 40/**89**/100 |
| sched 07 | 49/**78**/ 92 | 56/**82**/ 98 | 42/**75**/ 90 | 51/**77**/ 91 | 64/**78**/ 91 | 43/**68**/ 80 |
| sched 09 | 2/**24**/ 56 | 10/**31**/ 67 | 1/**22**/ 41 | 0/**18**/ 42 | 0/**21**/ 59 | 12/**28**/ 57 |
| sched 11 | 0/ **7**/ 16 | 4/ **8**/ 16 | 0/ **7**/ 23 | 0/ **6**/ 14 | 0/ **4**/ 11 | 4/ **8**/ 14 |
| t-tire 02 | 88/**95**/ 99 | 21/**80**/ 98 | 11/**70**/ 98 | 15/**69**/100 | 9/**77**/ 99 | 8/**60**/ 97 |
| t-tire 04 | 14/**56**/ 86 | 5/**49**/ 83 | 9/**56**/ 80 | 6/**53**/ 82 | 43/**61**/ 77 | 0/**31**/ 80 |
| t-tire 06 | 0/**17**/ 40 | 0/**26**/ 64 | 0/**30**/ 53 | 1/**25**/ 48 | 0/**18**/ 50 | 0/ **0**/ 1 |
| zeno 03 | 9/**65**/100 | 16/**78**/100 | 10/**45**/100 | 17/**54**/100 | 14/**55**/100 | 16/**77**/100 |
| zeno 05 | 11/**62**/100 | 10/**69**/100 | 7/**31**/100 | 9/**43**/ 99 | 12/**44**/ 99 | 8/**58**/100 |

Table 4: Minimum/average/maximum success rate.

sense. Comparing the two Blocksworld versions, we can make a speculation that is perhaps valid more generally:

*Rewards should be given conservatively.*

The two domains are similar, but Exploding-Blocks is more dangerous which is not reflected at the determinized level. The heuristics, in particular $h^{\mathsf{FF}}$, capture the basic structure fairly well and lead to good improvements in the basic domain, but not in the dangerous one. A possible explanation is that the heuristics provide too much flawed feedback to the learner (similar to, but less extreme than, what we observed for $h^{\mathsf{FF}}$ in Triangle-Tireworld). The goal counting is "conservative" in that it gives a reward only when part of the mission has actually been accomplished. In other words, it may be better to provide little feedback that is reliable, than to provide a lot of feedback that is noisy.

## Conclusion

Given our (speculative) observations regarding "conservative advice", we believe it would be interesting to investigate the impact of noisy advice in a clear-cut artificial setting. A hand-crafted search space could compare two heuristic functions, of which the conservative one gives correct advice but only in X% of the cases (in the other cases the heuristic value does not change), while the noisy heuristic gives advice all of the time but is wrong in Y% of the cases. It seems that this kind of setting could also lend itself to theoretical analysis.

The grand challenge remains to extend classical planning heuristics in a way so that they can take into account the probabilistic structure of the task at hand – this would be the only way to make them more conservative in domains like Exploding-Blocksworld and Triangle-Tireworld.

We finally remark that our analysis of the LAMA heuristic is a contribution in itself. The new versions of the heuristic would be worth trying out in classical planning. We note that the possible improvements don't stop at what we did for $h^{\mathsf{iLM}}$ herein. As mentioned, $<_n$ and $<_{gn}$ orders where the right hand side is initially true remain to be explored. Further, reasonable orders $l_1 <_r l_2$ allow an additional kind of reasoning not yet put to use: if $l_2$ is true and accepted, but $l_1$ is required again, then $l_2$ is required again, too; this can be applied transitively to literals ordered $<_r$ behind $l_2$. (Hence, in particular, $l_1 <_r l_2$ where both $l_1$ and $l_2$ are initially true may be useful for heuristic estimation.)

## References


Baxter, J.; Bartlett, P.; and Weaver, L. 2001. Experiments with infinite-horizon, policy-gradient estimation. *JAIR* 15:351–381.

Buffet, O., and Aberdeen, D. 2009. The factored policy-gradient planner. *AI* 173(5-6):722–747.

Dorigo, M., and Colombetti, M. 1994. Robot shaping: developing autonomous agents through learning. *AI* 71(2):321–370.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS'09*.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI'09*.

Matarić, M. 1994. Reward functions for accelerated learning. In *Proc. ICML'94*.

Ng, A.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. ICML'99*.

Randløv, J. 2000. Shaping in reinforcement learning by changing the physics of the problem. In *Proc. ICML'00*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI'08*, 975–982.

Szepesvári, C. 2009. Reinforcement learning algorithms for MDPs. Technical report.

Thiébaux, S.; Gretton, C.; Slaney, J.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-Markovian rewards. *JAIR* 25:17–74.

Wiewiora, E. 2003. Potential-based shaping and $Q$-value initialization are equivalent. *JAIR* 19:205–208.

Williams, R. 1992. Simple statistical gradient-following algorithms for connectionnist reinforcement learning. *ML* 8(3):229–256.

Yoon, S.; Fern, A.; and Givan, B. 2007. FF-Replan: a baseline for probabilistic planning. In *Proc. ICAPS'07*.