Planning Under Uncertainty Through Goal-Driven Action Selection

Juan Carlos Saborío[1(✉)] and Joachim Hertzberg[1,2]

[1] Institute of Computer Science, University of Osnabrück, Wachsbleiche 27, Osnabrück, Germany
jcsaborio@uos.de
[2] DFKI Robotics Innovation Center (Osnabrück), Albert-Einstein-Straße 1, Osnabrück, Germany

Abstract. Online planning in domains with uncertainty and partial observability conveys a series of performance challenges: agents must obtain information about the environment, quickly select actions with high reward prospects and avoid very expensive mistakes, while interleaving planning and execution in highly variable and uncertain domains. In order to reduce the amount of mistakes and help an agent focus on directly relevant actions, we propose a goal-driven, action selection method for planning in (PO)MDP's. This method introduces a reward bonus and a rollout policy for MCTS planners, both of which depend almost exclusively on a clear specification of the goal and produced promising results when planning in large domains of interest to cognitive and mobile robotics.

1 Introduction

Planning under uncertainty requires deliberating over actions, their effects, and computing values that reflect a combination of some form of utility or reward and their probability. Uncertainty in planning domains may come from non-deterministic actions or from incomplete knowledge about the environment and the agent's current state. These planning problems are often modelled as Markov Decision Processes (MDP's) or Partially Observable MDP's (POMDP's), and solved using many well known methods among which Monte-Carlo Tree Search (MCTS) is a popular choice, especially in the online planning community. UCT is the modern MCTS standard [1], and it guarantees asymptotic convergence and solutions that minimize regret by expanding a tree of states and selecting actions following the UCB1 formula [2]. An extension of UCT for partially observable domains, called POMCP [3], constitutes the (arguably) most general Monte-Carlo POMDP solver. As such POMCP is a basic starting point for online POMDP planning and the most obvious alternative to traditional, point-based POMDP solvers, most of which are simply incapable of solving moderately large problems.

We are interested in problems that can be modelled as POMDP's for several reasons: POMDP's explicitly represent the effect of information-gathering

J. van den Herik and A. P. Rocha (Eds.): ICAART 2018, LNAI 11352, pp. 182–201, 2019.
https://doi.org/10.1007/978-3-030-05453-3_9

actions, actions are assumed to be non-deterministic, and policies depend on a correct representation and estimation of the agent's true state. In other words, POMDP's correctly model the full extent of robotic task-planning in a mathematical framework with a strong analytical background. We are also interested in transferring these methods to planning onboard robots, but unlike many common AI problems, robot planning domains tend to be orders of magnitude more complex. Often, planning problems are reduced to their minimal and necessary elements and, while still potentially large, do not address the challenges that robots face when planning in the "real world": a massive amount of states that are reachable by the algorithm and yet, mostly irrelevant for any given goal. This means robot planning using POMDP's must follow a very strict, guided, goal-driven mechanism to avoid excessive computation.

The challenge to overcome in planning under uncertainty then becomes that of avoiding a very large number of states that contribute little to reaching the goal state, and identifying those that provide a significant contribution. This amounts to producing satisficing behavior, potentially overlooking parts of an optimal policy but generating visible results more quickly in very large planning domains. This approach responds to our attempts to provide a formal interpretation of the intuitive concept of "relevance", which in planning terms may be seen as a reliable (albeit imperfect) attentional filter guiding action selection, which may open up ways of handling problems with high dimensionality. Planning algorithms should be able to quickly identify promising (high expected value) states and focus on getting there. State values represent a weighted average of future rewards, so the problem reduces to quickly locating these sources of future rewards. One relatively simple idea is preferring actions that lead to subgoals (subsets of some terminal state) while avoiding those that don't, and encouraging these actions by providing additional, positive rewards. Achieving a subgoal objectively brings the agent a step closer to achieving a larger goal, and so we use this idea to formalize a metric of state-to-goal proximity.

We propose *partial goal satisfaction* as a way to compute the proximity of states to goals and provide a reward bonus in action selection, which easily becomes an action selection policy for Monte-Carlo rollouts. This is by no means a complete solution to online planning onboard robots, but rather a contribution towards the improvement of action selection in planning algorithms, when information about the goal is available. The effect is that the planning agent is encouraged to pursue certain promising actions, and receives optimistic value initializations in newly discovered states. This is a way of implicitly helping an agent or robot *do the right thing* by avoiding less promising alternatives during planning.

Existing approaches that address large planning spaces include value approximation and state aggregation, but these work under the assumption that there are large groups of states that can be clustered together (due to similarity or other reasons) using fixed criteria. At the moment we are interested in how agents may use knowledge of their goal(s) to improve their action selection criteria, in particular by focusing on only a few good alternatives when many options are

available, as is the case of domains with high variability and large branching factor.

In the following sections we discuss previous related work, and proceed to explain our proposal. We then provide a simple example in a fully observable MDP and two examples of large POMDP's, as well as an analysis of experimental results. We finalize by discussing the challenges of online POMDP planning and comment on future directions.

Please note that this is an extended version of a conference paper (see [4]). We introduce a new planning domain, test our proposed method in this new domain, and provide further analysis and considerations not present in our previous, related publication.

2 Notation

We rely on the standard notation for an MDP: let S and A be finite sets of states and actions respectively and $T(s, a, s') = P(s'|s, a)$ the probability of reaching state s' when executing a in s, which yields a real-valued reward $R(s, a, s')$. An MDP is the tuple $\langle S, A, T, R\rangle$, with discount factor γ.

In a POMDP, an agent receives an observation $\omega \in \Omega$ with probability $O(s, a, \omega) = P(\omega|s, a)$ and maintains an internal belief state $b \in B$, where b is a probability distribution over states and $b(s)$ is the probability of s being the current state. A POMDP is therefore $\langle S, A, T, R, \Omega, O\rangle$. The sequence $h_t = (a_0, \omega_1, \ldots, a_{t-1}, \omega_t)$ is the *history* at time t. Notice the complexity of planning in belief space, a $|S|$-dimensional hyperplane.

Many POMDP planning algorithms directly search over a tree of beliefs, explicitly reasoning about and choosing valuable (informative) beliefs, producing policies that are also given in terms of beliefs. We will however present our action selection bias in terms of states with mixed observability (states that contain both fully and partially observable features). This responds to two core principles: (1) Exploiting the structure of problems to simplify POMDP planning is possible and necessary, and in robotic task planning one simple and fair assumption is that there are some fully-observable features. (2) A state with fully-observable features can be sampled at any given point, using an approximator that provably approaches the true belief state, greatly reducing the complexity of planning.

3 Previous Work

POMDP planning has a very extensive literature that spans decades. Important highlights include the realization that value functions are piecewise-linear and convex (PWLC) and can thus be approximated by a PWLC function [5]. Algorithms such as Witness [6] or Incremental Pruning [7] actively select and discard vectors that correctly approximate the optimal value function. Anytime algorithms include HSVI [8], which follows heuristics derived from upper and lower bounds of the value function, PBVI [9], which carefully selects belief update

points and SARSOP [10], that avoids non-reachable beliefs. PBVI and SARSOP were actually tested in limited robotic applications or similar scenarios, but all of these algorithms are restricted to POMDP's so small, they fail to represent most robot planning scenarios.

Instead, POMCP follows a generative model approach through a POMDP simulator and an adapted version of UCT [3]. Its key contributions are approximating the current belief state using an unweighted particle filter, and expanding a tree of histories instead of a tree of states. This combination successfully addresses the *curse of dimensionality* and makes it possible to perform online planning in large POMDP's. Because this is a key improvement, this paper assumes a belief-state approximator and focuses on states with partially observable elements, instead of explicit belief states. The concept of mixed observability has produced positive results even outside of MCTS algorithms [11]. A similar Partially Observable UCT-based algorithm with more detailed belief selection also exists [12].

In order to address the state-space complexity of large POMDP's, well known techniques include clustering states and generalizing state or belief values [9, 13], function approximation [14] and random forest model learning [15]. These methods are based on fixed aggregation criteria that do not respond to the connection between states and goals, and despite (some of them) being anytime algorithms they still follow the slow belief tree search approach.

It is also possible to generate abstractions for planning and learning over hierarchies of actions [16,17]. This is inconvenient for general planning, as relatively detailed, prior knowledge of the domain is required to manually create these hierarchies. Recent work however shows a promising way to automatically construct action hierarchies [18].

Planning algorithms for MDP's and POMDP's often overlap with reinforcement learning (RL) methods, with the difference that in RL the agent must find an optimal policy while discovering and learning the transition dynamics. Reward shaping is commonly used in RL to improve an agent's performance by awarding additional rewards for certain preferred actions, implicitly defining subgoals. This generates a decision process with a different reward distribution and therefore different convergence properties, but potential-based reward shaping (PBRS) has been shown to preserve policy optimality [19]. A study of PBRS in the context of online POMDP planning can be found in [20].

Building on these arguments, this paper reflects our efforts to provide a general-purpose, PBRS bias for action selection under uncertainty, in order to address the complexity of planning in large domains using only partial information, as is the case of robotics.

4 Measuring Goal Proximity

An ideally efficient planning algorithm should quickly separate *good* or promising states from *bad* or unwanted states. In other words, it should quickly prefer states that lead to the goal and avoid a large number of those that don't. Most

planning domains, even those without clearly specified goals (eg. pure RL tasks), have terminal states or conditions that specify what must be accomplished and, to some extent, what subgoals the agent should pursue. In robotics, it is reasonable to assume planning agents are somewhat informed and aware of at least part of their goal(s). Any sufficiently detailed state description (such as a feature vector) provides information to compute, for any given state, some numerical score representing how many features in the terminal state have already been accomplished. The larger this number is, the closer this state is to being a terminal or a goal state. We call this idea *partial goal satisfaction* (PGS), formalized in Eq. 1. A previous version of this section, including equations, was published in [4].

PGS is simple to implement for fully observable features, which can be easily counted in meaningful ways (Eq. 2). For partially observable features, information gathering actions should increase the probability that their current, estimated value is correct, thus also affecting the probability of an agent being in some given state ($b(s)$). In other words, collecting information about a given set of partially-observable features yields a better estimate of the world's current, true state. The simplest, most general approach is therefore measuring some form of uncertainty or entropy and providing rewards as this uncertainty is reduced (Eq. 3). Let $s \in S$ be a state, decomposed into countable discrete features s_i, G_+ be the set containing the observable features present in the goal, G_- the set of observable restrictions, $\Delta(s)$ the set of states reachable from s (similar to the transitive closure of $T(s, \cdot)$) and G_p the set of partially or non-observable elements, then:

$$\text{pgs}(s) = \sum_{s_i \notin G_p} v(s_i) + \sum_{s_j \in G_p} w(s_j) \tag{1}$$

where:

$$v(s_i) = \begin{cases} 1 & \text{iff } s_i \in G_+ \\ x \in (0,1) & \text{iff } \exists s' \in \Delta(s) \text{ s.t.:} \\ & s'_k \in s' \wedge s'_k \in G_+ \\ 0 & \text{iff } s_i \notin \{G_+ \cup G_-\} \\ -1 & \text{iff } s_i \in G_- \end{cases} \tag{2}$$

and

$$w(s_j) = \begin{cases} 0 & \text{iff } H(s_j) \leqslant T_H \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

This means the different features in each state are evaluated depending on whether they are partially observable ($s_j \in G_p$) or not. Positive, observable features add points and negative features deduct points. State changes that lead to a positive feature ($s'_k \in G_+$) in a future state ($s' \in \Delta(s)$) yield a fraction of a point and help implicitly define subgoals (eg. interacting with an object referenced in the goal, such as picking up the coffee cup that goes on the table), and if no relevant features are present no points are awarded or taken. Partially-observable features are scored based on the entropy of their underlying distribution, punishing features or states with high entropy. Whenever enough information is

gathered and the entropy is reduced below some threshold T_H, this punishment is removed. This encourages the agent to quickly get rid of this penalty by executing a number of information gathering actions, which in turn may lead to discovering new reward sources (eg. interacting with relevant but previously unrecognized elements). In principle any combination of the individual elements in the goal may be considered a subgoal for scoring purposes, and only completing all of them simultaneously yields the total, problem-defined terminal reward. This scoring is derived from a clearly specified goal, which should always available to a planning agent or robot. Some very specific problems, however, might also benefit from introducing some amount of domain information.

PGS may be useful in different contexts, but it is intended as an optimistic value initialization method that allows an agent to identify subgoals and exploit immediate opportunities if available. As such, it addresses action selection under uncertainty, not the full planning problem. Using PGS directly to solve classical planning problems, such as some Blocks World configuration, may result in overly greedy actions. As explained in the next subsections, PGS is intended to be used as a reward bonus in planning algorithms, and as a rollout policy in the context of Monte-Carlo or similar planning algorithms, where optimistic assumptions will eventually be corrected (if they're wrong) and the problem solved properly.

4.1 PGS in Reward Shaping

Reward shaping is a well-known technique used to improve the performance of (PO)MDP algorithms and RL problems. It works by adding a small, additional reward to some state transitions, encouraging the agent to choose certain *implicitly* preferred actions. In practice, the amount of reward bonus often comes from an in-depth analysis of the structure of the problem and provides some form of heuristic bias in action selection. In our case, instead of providing explicit, domain-dependent knowledge to shape rewards, we use the PGS function to encourage the agent to pursue courses of action leading to the completion of subgoals. The reward bonus produces a new reward distribution and therefore a potentially different problem, one with additional reward sources. Following the PBRS form however, guarantees the introduction of implicit subgoals doesn't affect the algorithm's convergence and, therefore, policies are transferable to the original problem.

Reward shaping substitutes the usual reward function in an MDP with:

$$R(s, a, s') + F(s, a, s') \tag{4}$$

where R is the problem-defined reward distribution and F is a reward bonus. If F has the form

$$F(s, a, s') = \gamma\phi(s') - \phi(s) \tag{5}$$

then it is a potential function and Eq. 4 is potential-based. We now define $\phi(s)$ for PGS as

$$\phi(s) = \alpha \mathrm{pgs}(s) \tag{6}$$

where α a scaling factor. Because most (PO)MDP algorithms already use γ to refer to the discount factor, from now on we will refer to γ_{PGS} when in the context of PBRS. In practice, transitions to states that are *closer* to a subgoal (positive reward source) will produce a positive difference, transitions to states that are farther from subgoals generate a negative difference, and other transitions cancel each other out. Normally reward shaping functions are highly specific for particular problems, but PGS manages to attain simplicity and generality.

4.2 PGS as a Rollout Policy

Monte-Carlo Tree Search algorithms, such as UCT and POMCP, work by sampling sequences of states from a probabilistic transition model. A tree of states (or in the case of POMCP a tree of histories) is progressively expanded and the average returns and visit counts are maintained per tree node. When enough statistics are available (eg. all known successors of a state have been visited) the UCB1 rule is used to select an action. When a new state is discovered, a rollout or random simulation is performed and its outcome used as an initial value estimation. Rollout policies are therefore largely responsible for the performance of MCTS online planning algorithms. Using PGS as a rollout policy, the agent quickly focuses on actions that directly contribute to the completion of (sub)goals and, likewise, avoids undesirable actions. Selecting actions that maximize state-to-goal proximity can implicitly summarize a very rich array of knowledge and heuristics, that must otherwise be given explicitly. To the best of our knowledge, the effect of evaluating goal proximity within the context of Monte-Carlo rollouts hasn't yet been systematically studied.

Using PGS as a rollout policy is very simple: Let s be the current state and $\mathcal{A}$ a set of actions. Then select the action $a \in \mathcal{A}$ leading to the state $s' \leftarrow (s, a)$ that satisfies the largest amount of subgoals, where ties are broken randomly. The action selection policy is formalized in Eq. 7:

$$\mathcal{A}(s) = \arg\max_{a} \text{pgs}(s' \leftarrow (s, a)) \tag{7}$$

In line with the goal-based approach, $\mathcal{A}$ could be defined as the action set consisting of legal actions and uncertainty reducing actions, avoiding information gathering actions if their effects do not provide more information (eg. if feature $j \in s$ affected by such action already satisfies $H(s_j) \leqslant T_H$). For example, avoid action "*scan object* 3" during rollouts if there is enough information to assume it is a cup.

Because PGS is computed as a difference between the current and previous states (Eq. 5), when $\gamma_{\text{PGS}} = 1$ only newly completed subgoals produce positive values. For example imagine a robot tasked with collecting and delivering a cup of coffee: during planning, standing next to the cup offers the possibility of picking it up, satisfying a subgoal that yields a reward bonus, therefore becoming the preferred action of the rollout policy. Once holding it, dropping the cup in any place other than the correct location reverts this condition and produces a negative reward, meaning it will never be chosen in a rollout (albeit eventually

during simulation, if all actions are systematically sampled). Online Monte-Carlo planning produces an action recommendation only after arbitrarily many simulations have been carried out, but starting out with the (seemingly) right action greatly improves performance. Unlike with PGS, improved rollout policies often rely on manually designed heuristics and explicit preferred actions.

5 Results

We tested PGS in two well-known and commonly used benchmark problems, as well as a new problem introduced in this paper. The first one is known as the "Taxi domain", and it defines a fully observable MDP useful to test basic functionality. The second is Rocksample, a POMDP that can be scaled up to fairly large state spaces. The third problem, explained in more detail below, is called Cellar and is a derivation of Rocksample with additional objects, observations and a different reward distribution, in an attempt to more closely resemble robotic planning. For the taxi problem we implemented our own version of UCT, and for the last two we modified the POMCP source code. All tests ran on a desktop workstation with an Intel i7-4790 CPU, 20 GB RAM and Debian GNU/Linux, and both planners are programmed in C++.

The challenge for robot planning under uncertainty is achieving good performance within a finite horizon, fast enough, even in large problems. These scenarios show the performance of PGS using limited resources (very few or relatively few Monte-Carlo simulations) and how it scales in considerably large POMDP's.

Some of the results presented in this section were previously published in [4]. Subsection 5.1, however, contains additional experiments and Subsect. 5.3 (Cellar domain) is completely new.

5.1 Taxi Domain

The taxi domain, first proposed in [17], is a simple, fully-observable MDP often used to test planning and learning algorithms. The taxi agent moves in any of four directions in a 5×5 grid and must pick up a passenger located in one of four possible depots, and bring it to another depot. A slight variation is the "fickle taxi" in which movement is non-deterministic: with a small probability (eg. $p = 0.1$) the taxi will end up East or West of its intended direction. Possible actions are moving North, East, South or West, collecting a passenger when standing on the same grid cell or dropping the passenger (when carrying one). Rewards are -1 for each regular move, -10 for dropping the passenger in the wrong location and 20 for delivering a passenger correctly, which also terminates the episode. We chose one instance of the taxi domain and obtained the total discounted reward of its optimal policy, 8.652 (with $\gamma = 0.95$), in order to compare it with the experimental results. This particular configuration and in general the taxi domain are illustrated in Fig. 1, where the dark cell at the top left corner is the goal depot where the passenger must be dropped. The walls shown in the

picture are also included in the experiment which means the agent's movement is restricted, in cells next to walls, to only open, adjacent cells.

Fig. 1. The taxi domain. Source: [4].

Because all elements are fully observable, PGS in the taxi domain is easy to formalize. We simply award 0.25 points for picking up the passenger and 1 point for dropping it at the correct depot (G_+). There are no restrictions ($G_- = \emptyset$). The terminal state reward is preserved but PGS reward substitution is used for the rest, with $\gamma_{\text{PGS}} = 1$ and $\alpha = 10$ to reflect the punishment for illegally dropping a passenger. Finally, a discount factor of $\gamma = 0.95$ and search depth of 90 steps were used within UCT. It is common to allow the taxi to start only over a depot, but in our experiments it could be anywhere on the grid. We ran UCT with PGS in both (regular and fickle) versions of the taxi domain, and obtained the average discounted rewards and running times over 1000 runs. Table 1 shows the result of repeated runs on the fixed task (Fig. 1) and a set of randomly generated episodes (randomized origin and destination depots, and taxi starting location) using 1024 simulations, extremely few for Monte-Carlo standards.

Table 1. Performance in the taxi domain with 1024 simulations. Source: [4].

Transition	Episodes	Avg. return	Time
Normal	Fixed	6.161	3.049
	Random	4.257	3.531
Fickle	Fixed	3.275	4.410
	Random	2.138	4.176

Results are promising if we compare the average discounted reward with the optimal policy in the fixed (non fickle) task (8.652). Restricting the amount of computation to only 256 simulations per move (≈1.6 s. per episode), the PGS-based planner achieved an average discounted reward of 5.089. On random tasks

it is important to mention that episodes were terminated after 5 s., but their (negative) reward still averaged.

Averaging performance, especially in stochastic domains, may hide interesting details of particularly good runs. We ran a separate batch of 1000 episodes using 1024 simulations, of which 616 finished in 2 s. or less and 797 in 3 s. or less. In these test the statistical mode was the maximum discounted reward (8.652), meaning most runs found the optimal policy.

Finally, Fig. 2 shows how the PGS method scaled in the Taxi domain. Performance was averaged over 1000 randomly generated runs, with a more strict acting budget of 35 steps but a more generous timeout per episode of 30 s. to allow for enough planning time with up to 8192 simulations. PGS quickly achieved satisfactory performance, even with very few simulations which translates into a very short planning time (shown at the top). We don't include comparative results with plain UCT (without PGS) because, in practice, it required (comparatively) excessive amounts of time and simulations to achieve even little improvements in performance.

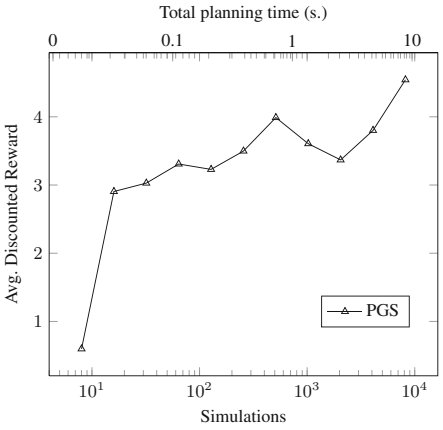


Fig. 2. Performance in the Taxi domain.

These results show how performance can be substantially improved in fully observable tasks by following a goal-driven, action selection method that implicitly exploits the problem structure. In the following subsections, we present results in problems with partial observability.

5.2 Rocksample

Rocksample, originally found in [8], is a commonly used problem that roughly simulates a Mars rover tasked with collecting valuable rocks. This problem corresponds to a POMDP in which the location of the agent and the rocks are known, but the value of these rocks is initially unknown and must be determined by the

use of a noisy sensor that returns one of two observations, *good* or *bad*, with a given reliability. *Rocksample*$[n, k]$ defines an $n \times n$ grid with k rocks, where the agent may move in any of four directions, sample a rock if standing directly on top of it, or use the sensor on any rock (action $check_i$ for rock i) for a total of $5 + k$ actions (see Fig. 3). Rewards are 10 for sampling good rocks, -10 for sampling bad rocks, 10 for exiting (East) and -100 for leaving the grid in any other direction [8]. We used POMCP as a POMDP solver [3], but modified it to test our proposal.

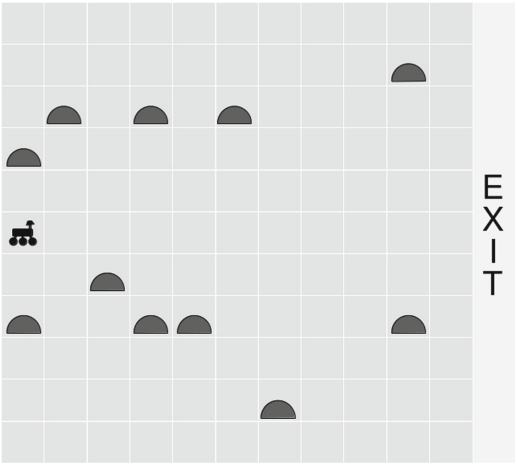


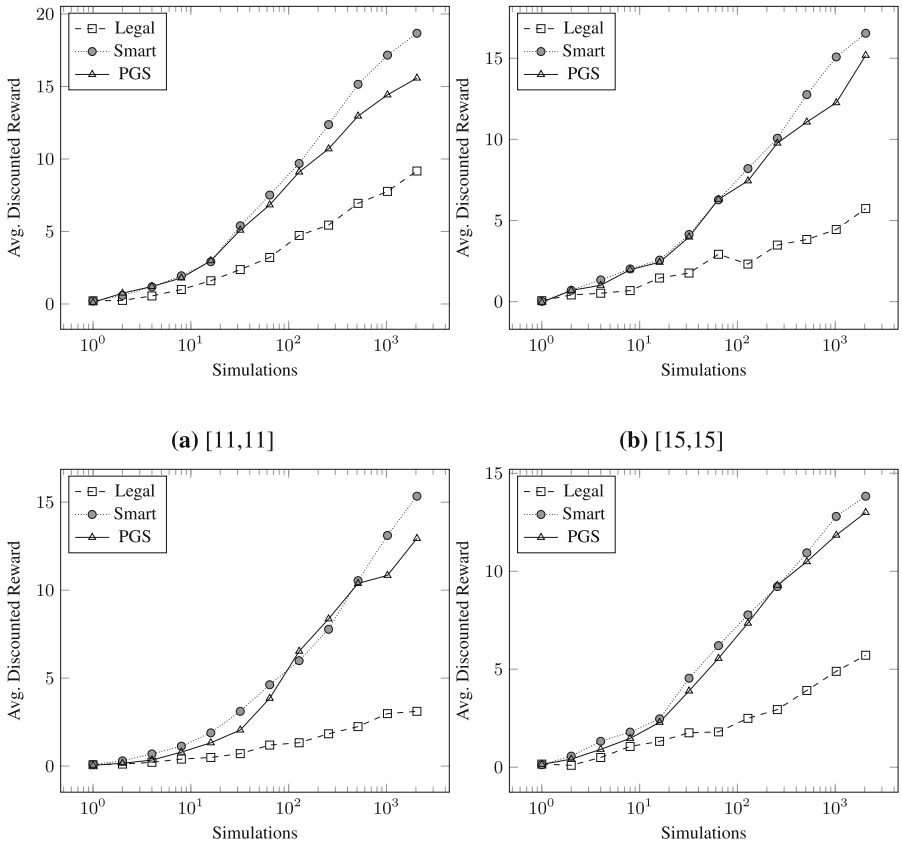
Fig. 3. Special layout for Rocksample[11, 11]. Source: [4].

POMCP uses slightly enhanced rocksample states, where the probability that a rock is good is updated directly after every corresponding *check* action, using the sensor efficiency and the previous likelihood. We defined $C = G_+ \cap G_-$ to be the set of collected rocks, and G_p the remaining rocks. Scoring function $v(s_i)$ returns 1 for good rocks with good observations (G_+) and -1 for bad rocks (G_-). Function $w(s_j)$ returns -1 if $H_b(p_r) > 0.5$, that is, if the binary entropy of rock r (s_j) is higher than 0.5. POMCP comes with a preferred actions policy, which uses manually encoded heuristics such as "head North if there are rocks with more positive observations" or "check rocks that have been measured less than five times and have less than two positive observations". Clearly, PGS succeeds in avoiding this level of over specification.

We used $\gamma_{\text{PGS}} = 1$ and $\alpha = 10$ (to reflect the difference in rewards received when sampling good and bad rocks). This scoring function deducts points for undesirable states (eg. collecting bad rocks, high entropy for any rock) and only adds points when collecting good rocks, but further negative points are withdrawn once the knowledge about any particular rock increases (i.e. entropy < 0.5). In practice this means that during rollouts *check* will be preferred if it

reduces entropy for some rock, that *sample* will be preferred when standing over a promising rock, and that otherwise movement actions will be considered.

We compared three different policies: uniformly random with legal moves ("legal" in POMCP), explicit preferred actions ("smart" in POMCP) and our own, "PGS". Figure 4 shows the discounted rewards averaged over 1000 runs for all three policies in *rocksample* [11, 11], [15, 15], and the large [25, 25] and [12, 25], with up to 2048 Monte-Carlo simulations per move.



(a) [11,11] **(b)** [15,15]

(c) [25,25] **(d)** [25,12]

Fig. 4. Performance in Rocksample. Source: [4].

PGS clearly outperforms the legal policy and is only slightly outmatched by the smart policy. This difference however reduces as the problem size increases, particularly in [25, 25], a very large problem for *rocksample* standards and in [25, 12], an equally large grid but with fewer reward sources. Estimating the PGS value of derived states for every action in our rollout policy may be somewhat

computationally expensive, particularly when all or many actions are available, since it trades off simulations for runtime. Results show, however, this pays off compared to the random policy: with ≈ 2.5 s of computation in *rocksample*$[25, 12]$, the legal rollout policy achieves a discounted reward of 5.713 with 2048 simulations, whereas PGS collects 7.35 with only 128 simulations. A faster implementation for planning onboard robots might be necessary, but also a shallower planning depth may be used (the experiments used a depth of 90 steps).

5.3 Cellar

We will now present the Cellar domain, devised as a series of POMDP's that more closely model robot task-planning under uncertainty. Problems commonly used in planning, including Rocksample, suffer from several oversimplifications: they represent only the elements or features that are directly relevant to the goal, and make unrealistic assumptions about the cost of actions such of moving and scanning or checking. While Rocksample succeeds at representing the goals of planning, it does not represent important challenges: in the "real world", a robot finds and interacts with many different objects, it has many possible actions and it receives many different observations. From these only a few are actually relevant to achieve the goal, and the rest are obstacles that must nonetheless be addressed one way or another while planning.

In this problem, clearly inspired by Rocksample, the agent must navigate a wine cellar and collect *at least one* valuable bottle, while it encounters objects such as shelves and crates that may or may not help to reach the goal. Cellar$[n, k, s, c]$ defines an $n \times n$ grid, with k bottles, s shelves and c crates. Crates can be pushed in the direction of any free grid cell (i.e. no bottles or objects), but attempting to push anything else is (more heavily) punished. A *check* action is always available for any object and any bottle, and the agent may move in four possible directions (North, West, South, East) resulting in a total of $9 + k + s + c$ actions. For simplicity, each bottle can be either *good* or *bad* and each object can be either a crate or a shelf, resulting in 4 total observations but each pair exclusive to their object class. Initially the agent knows the location of the bottles and of the objects, the bottles have equal probability of being good or bad and the objects equal probability of being a crate or a shelf. The sensor efficiency and the derived *check* actions work exactly like in Rocksample.

We designed special layouts for two cases of interest: Cellar[7, 8, 7, 8] and Cellar[11, 11, 15, 15]. The first is a POMDP with 32 actions and 2×2 observations resulting in a state space of more than 10^{15} states. The second problem defines a POMDP with 50 actions, 2×2 observations, and a seriously large state space: approximately 10^{31}. In comparison, Rocksample[11, 11] has $247,808$ states and Rocksample[25, 25], approximately 10^{10} states. The larger Cellar problem is illustrated in Fig. 5, where the tall rectangles represent shelves, the short squares represent crates, and the bottles are shown in dark red.

The reward distribution for the Cellar problem is $+10$ for collecting a good bottle, -10 for collecting a bad bottle, $+10$ for the terminal state (leaving the

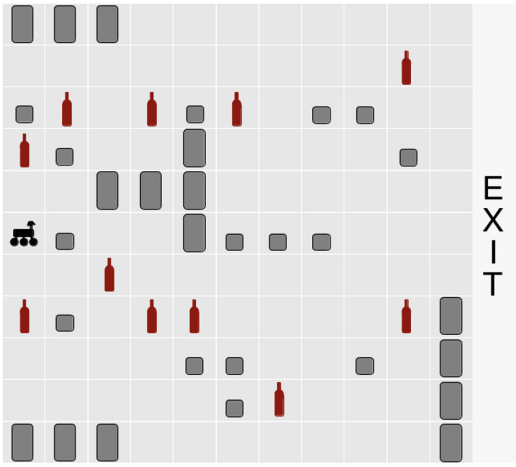


Fig. 5. Layout of Cellar[11, 11, 15, 15]. (Color figure online)

grid to the East with at least one good bottle), −0.5 for checking, −1 per movement step, −2 for pushing crates and −10 for pushing anything else. This distribution reflects our attempts to better capture robot planning, where no action is truly free and some actions (such as pushing) can be relatively expensive even if they are necessary. Additionally, rewards implicitly relate to the goal and in domains as complex as this one, time restrictions with free actions can lead to undesirable (yet rational) behavior such as spending the allotted time performing only *check* actions and not moving or sampling, thus minimizing the loss in the total cumulative reward but not actually reaching a terminal state. This type of behavior may result in very poor policies with deceivingly acceptable performance (not much reward loss), a detail lost when presenting only average performance tables.

PGS scoring in Cellar is equivalent to its Rocksample counterpart: collecting valuable or non-valuable bottles yields its respective fully-observable points, and uncertainty about bottles yields equivalent punishments. Uncertainty about objects such as shelves and crates should not be punished however, because they do not form part of the goal. Rewarding (or lifting the punishment) for *checking* objects encourages the agent to acquire potentially unnecessary knowledge and to execute many, relatively expensive actions. An interesting challenge, to be addressed in future work, is deciding when to gather information about surrounding objects. That is, identifying which non goal-related, information-gathering actions actually contribute to increasing the total reward.

Unlike in Rocksample, however, during rollouts we consider only movement, sampling, pushing and uncertainty-reducing *checks*, meaning we don't check objects that already meet the entropy requirements. We adapted the preferred actions in the "Smart" Rocksample policy (included with POMCP) so that the same movement, checking and sampling heuristics apply, and added equivalent

heuristics for object checking. Likewise, "Legal" refers to the uniformly random policy that considers all valid actions (eg. not leaving the grid).

In order to stress the importance of quickly *doing the right thing*, let us first consider a minimal version of this problem, cellar[5, 1, 0, 4], where four crates surround a single, valuable bottle (Fig. 6). This relatively straightforward POMDP has a very clear, recognizable goal, and yet it has around 6 million states. Agents solving this problem should quickly realize they must push a crate, collect the bottle and leave, trying not to move around aimlessly or unnecessarily checking and pushing.

Fig. 6. Minimal Cellar example.

Figure 7 shows the performance of all three policies in the minimal version of the Cellar domain. The total discounted return was averaged over 100 runs with up to 2048 Monte-Carlo simulations per step and discount $\gamma = 0.99$, t. In this problem, after 128 simulations PGS dramatically outperformed the two other action selection policies. Out of the 100 runs, PGS reached the terminal state in 97 of them with only 512 simulations, and with 1024 simulations or more, 99 runs reached the terminal state. Non-terminal runs ended when the agent spent its allotted *acting* budget, set to a generous 100 steps. The heuristic policy was capable of terminating a maximum of 64 times, whereas the random policy a maximum of only 37 despite having a similar mean discounted reward. In this configuration the low bar set by the terminal state is actually informative given that there is exactly one bottle to collect.

The comparative performance of PGS in the larger Cellar problems is shown in Fig. 8, averaged over 100 runs with up to 8192 Monte-Carlo simulations per move, with a large acting budget of 500 steps. In cellar[7, 8, 7, 8] we used a discount factor of $\gamma = 0.99$, but chose a discount factor of $\gamma = 0.95$ in cellar[11, 11, 15, 15] in order to make the problem slightly more manageable by reducing the search horizon (which also results in a different range of rewards).

These results in the full Cellar problems show how PGS scaled in very large, task-planning POMDP's. In the first case (Fig. 8a) with very few simulations PGS already performed slightly better than two other policies, and with more simulations it clearly outperformed the random and the heuristic policies by a fair margin. Interestingly, PGS was also faster than Smart: in less than half the

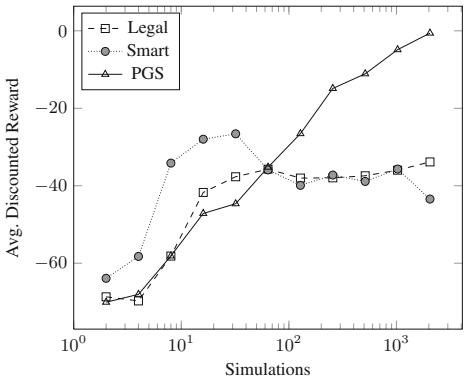
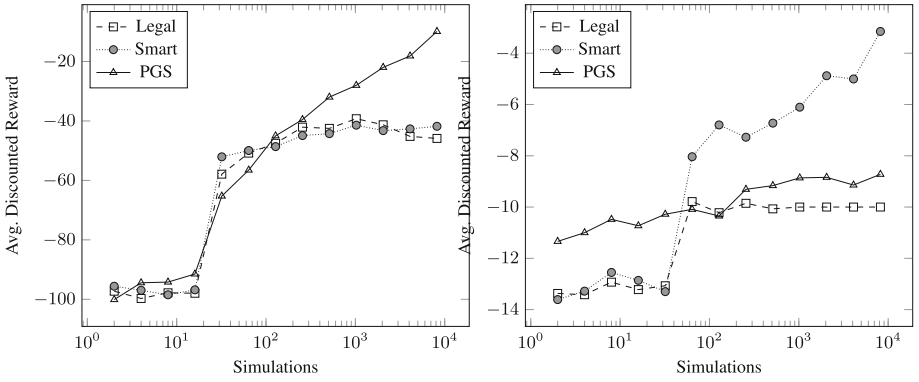


Fig. 7. Performance in cellar[5, 1, 0, 4].



(a) [7,8,7,8] **(b)** [11,11,15,15]

Fig. 8. Performance in the Cellar domain.

time, it gathered more discounted reward and achieved a much higher terminal state count.

In the second case (Fig. 8b), PGS improved upon the overall performance achieved by the random policy and, with very few simulations, achieved comparatively better performance than both competing policies. However, due to the extremely large state-space and branching factor of this POMDP, it didn't scale as well as the manually designed, heuristic policy. The Legal policy, on the other hand, quickly settled on a low, local maximum performance possibly limited by its uninformed, uniformly random sampling. It must be noted that due to the size of the second problem there's little difference in the total discounted reward across the three policies, up to the maximum amount of simulations used, which suggests we might essentially be looking at the beginning of the plot. Perhaps if significantly more simulations were used, sizable differences in performance

would appear, but that amount of computational resources and response times are already outside the scope of *online* planning.

In order to transfer these methods to robots we must focus on quickly achieving a significant increase in performance, since we're attempting to minimize response time (and computing resources). This type of reasonably satisfactory behavior with relatively few simulations is precisely what should be expected from a more informed action-selection policy. Efficiently solving large task-planning problems, such as cellar[11, 11, 15, 15] and beyond, requires more than "just" better action selection. For the purpose of this paper, the Cellar domain allowed us to measure the scalability of our goal-driven bias in very complex scenarios, but it will be used again in the future in the context of managing state complexity.

6 Conclusions and Discussion

Our experimental results show that despite its simplicity, PGS effectively improves the performance of planning in large (PO)MDP's, thanks to its goal-driven approach to action selection. In the fully observable problem (Taxi domain), PGS achieved a level of performance far out of reach for a uniformly random policy in standard UCT.

In domains with partial and mixed observability and particularly in problems with scarce reward sources, such as larger versions of Rocksample, PGS easily outperformed the uniformly random policy and closely followed a manually designed, heuristic policy. In such problems, the random policy scaled poorly and domain knowledge became necessary to achieve good performance quickly. We showed, however, that with barely any domain-dependent knowledge, PGS can be competitive with a manually designed action-selection policy even if it relies on detailed, heuristic knowledge. This type of domain-independent bias is essential for planning and acting in complex domains, and estimating the information gain of uncertainty-reducing actions may be necessary to correctly address large robotic planning domains, avoiding less useful choices.

The second partially observable domain, Cellar, showed the performance of a goal-driven bias in action selection when planning in very large domains with obstacles, unnecessary actions and a more realistic reward distribution. The minimalistic version of this domain underlines the importance of quickly identifying reward sources and choosing actions that lead to them, which both the uniform and heuristic policies failed to do. The larger problems constitute a much more complicated optimization challenge that nonetheless benefits from goal-oriented behavior, as shown by the performance of PGS in cellar[7, 8, 7, 8]. As mentioned in the previous section, the largest of these problems requires more refined mechanisms that directly address the state complexity. In principle for any given planning problem it is possible to design a detailed and extremely efficient heuristic rollout policy using domain knowledge, in the spirit of "Smart", and it should perform and scale relatively well. However general-purpose planning under uncertainty, especially onboard robots, cannot rely on

manually designed, domain-dependent heuristics. When solving practical, well-understood problems in controlled scenarios, the combination of both domain-knowledge and goal-driven action selection might produce very promising results.

In general we can identify three main approaches for speeding up planning in large stochastic domains: (1) Action hierarchies that produce smaller, abstract MDP's and then transfer these solutions to the base MDP. a minimal version of (2) State abstractions that group states together so their values are shared and the values of unknown states, approximated. a minimal version of (3) PBRS, that forces an agent to focus on good action prospects, avoiding potentially costly choices. Current techniques for both action and state abstraction rely on fixed criteria that might cause a planner to traverse many unique states anyway, which is the challenge addressed by PGS. We attempt to quickly identify reward sources and back propagate scaled partial rewards, using as little domain knowledge as possible but exploiting an agent's knowledge of its own goals.

As previously stated, this work is only part of our efforts to introduce the notion of relevance in task planning. Future work includes designing *dynamic* value approximation and/or state aggregation methods derived from this methodology, as well as limiting the amount of reachable states to those that directly contribute to the goal. In order to transfer these methods to real-world robotic tasks, it will also be necessary to map continuous to discrete state representation. We argue that similar to PGS, dimensionality reduction techniques should consider criteria derived from the goal (something that, as far as we know, hasn't been tried yet).

In order to obtain the results presented in this paper we experimented extensively on the three domains, using different parameters and observing different types of behavior when such parameters were modified. It is very well known that UCT is sensitive to certain important variables, such as the number of simulations per step, the search horizon or discount factor, and the exploration rate in UCB1 action selection. The really significant leap in performance for any kind of planning agent (human, program or robot) however, comes from an appropriate modeling or understanding of the domain, including available actions and goals. In problems such as cellar[5, 1, 0, 4] we can as humans intuitively understand, very quickly, what needs to be done and promptly suggest a sufficiently good plan (maybe even optimal), despite the almost 6 million possible states and the much, much larger belief space. Similar problems in larger grids (eg. imagine cellar[50, 1, 0, 4]) have exponentially larger state and belief spaces (approximately 10^{17} states) and yet the problem structure is essentially the same. The agent must "simply" focus on a few good states and actions, and avoid the rest (large empty grids can in fact be simplified with state aggregation and action hierarchies). It seems, then, that efficient planning requires more than just evaluating many reachable states and beliefs.

Often the perceived complexity of planning problems is justified based on their worst-case computational complexity, which is a direct function of the number of states. Planning problems however are defined not only in terms of their transition dynamics, but also in terms of some implicit or explicit goal,

so the intrinsic relationship between the values of states and their proximity to the goal (or subgoals) must be considered. An efficient planning algorithm should quickly identify the gaps in the different state values, in order to separate good states from bad states, and focus on the good ones. These values come uniquely from perceived (or simulated) rewards. Because the value of a state is the average discounted return of its children, it might be difficult to differentiate promising states from less promising ones early on, when the agent is simply too far (in terms of state or belief transitions) from any source of reward and doesn't have enough knowledge or statistics to decide. However, if an agent finds itself initially "close" to its goal or a reward source (eg. a few actions away), the problem should be simple to address or even solve if the appropriate actions are chosen, regardless of the number of reachable states and beliefs.

We designed PGS, a simple and straightforward action selection bias, as an attempt to exploit these observations. Experimental results indicate that it performs and scales well even in large domains, and underline the importance of goal-oriented behavior. We expect this type of relevance-driven methodology to improve the performance of online planning under uncertainty in robots and similar agents, that must quickly act with only limited information despite the complexity of "real-world" problems.

Acknowledgements. We would like to thank our colleagues Sebastian Pütz and Felix Igelbrink for their suggested reward distribution in the Cellar domain, and the DAAD for supporting this work with a research grant.

References

1. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006). https://doi.org/10.1007/11871842_29
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**, 235–256 (2002)
3. Silver, D., Veness, J.: Monte-Carlo planning in large POMDPs. Adv. Neural Inf. Process. Syst. **23**, 2164–2172 (2010)
4. Saborío, J.C., Hertzberg, J.: Towards domain-independent biases for action selection in robotic task-planning under uncertainty. In: Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART, INSTICC, vol. 2, pp. 85–93. SciTePress (2018)
5. Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable Markov processes over a finite Horizon. Oper. Res. **21**, 1071–1088 (1973)
6. Cassandra, A.R., Kaelbling, L.P., Littman, M.L.: Acting optimally in partially observable stochastic domains. In: Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, 31 July – 4 August, vol. 2, pp. 1023–1028 (1994)
7. Cassandra, A.R., Littman, M.L., Zhang, N.L.: Incremental pruning: a simple, fast, exact method for partially observable markov decision processes. In: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, UAI 1997, Brown University, Providence, Rhode Island, USA, 1–3 August 1997, pp. 54–61 (1997)

8. Smith, T., Simmons, R.: Heuristic search value iteration for POMDPs. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, UAI 2004, pp. 520–527. AUAI Press, Arlington (2004)
9. Pineau, J., Gordon, G.J., Thrun, S.: Anytime point-based approximations for large POMDPs. J. Artif. Intell. Res. **27**, 335–380 (2006)
10. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems IV, Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, 25–28 June 2008 (2008)
11. Ong, S.C.W., Png, S.W., Hsu, D., Lee, W.S.: Planning under uncertainty for robotic tasks with mixed observability. Int. J. Rob. Res. **29**, 1053–1068 (2010)
12. Somani, A., Ye, N., Hsu, D., Lee, W.S.: DESPOT: online POMDP planning with regularization. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 26, pp. 1772–1780. Curran Associates, Inc. (2013)
13. Pineau, J., Gordon, G., Thrun, S.: Policy-contingent abstraction for robust robot control. In: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, UAI 2003, pp. 477–484. Morgan Kaufmann Publishers Inc., San Francisco (2003)
14. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. MIT Press, Cambridge (2012). (to be published)
15. Hester, T., Stone, P.: TEXPLORE: real-time sample-efficient reinforcement learning for robots. Mach. Learn. **90**, 385–429 (2013)
16. Sutton, R., Precup, D., Singh, S.: Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**, 181–211 (1999)
17. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. **13**, 227–303 (2000)
18. Konidaris, G.: Constructing abstraction hierarchies using a skill-symbol loop. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp. 1648–1654 (2016)
19. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning, pp. 278–287. Morgan Kaufmann (1999)
20. Eck, A., Soh, L.K., Devlin, S., Kudenko, D.: Potential-based reward shaping for finite Horizon online POMDP planning. Auton. Agents Multi-agent Syst. **30**, 403–445 (2016)