Annals of Mathematics and Artificial Intelligence
https://doi.org/10.1007/s10472-024-09947-5

RESEARCH

Domain independent heuristics for online stochastic contingent planning

Oded Blumenthal[1] · **Guy Shani**[1]

Accepted: 12 May 2024

Abstract
Partially observable Markov decision processes (POMDP) are a useful model for decision-making under partial observability and stochastic actions. Partially Observable Monte-Carlo Planning (POMCP) is an online algorithm for deciding on the next action to perform, using a Monte-Carlo tree search approach, based on the UCT algorithm for fully observable Markov-decision processes. POMCP develops an action-observation tree, and at the leaves, uses a rollout policy to provide a value estimate for the leaf. As such, POMCP is highly dependent on the rollout policy to compute good estimates, and hence identify good actions. Thus, many practitioners who use POMCP are required to create strong, domain-specific heuristics. In this paper, we model POMDPs as stochastic contingent planning problems. This allows us to leverage domain-independent heuristics that were developed in the planning community. We suggest two heuristics, the first is based on the well-known h_{add} heuristic from classical planning, and the second is computed in belief space, taking the value of information into account.

Keywords POMDP · Contingent planning · Heuristics · Online planning

Mathematics Subject Classification (2010) 68T20

1 Introduction

Many autonomous agents operate in environments where actions have stochastic effects, and important information that is required for obtaining the goal is hidden from the agent. Agents in such environments typically execute actions and sense some observations that result from these actions. Based on the accumulated observations the agents can better estimate their current state and decide on the next action to execute. Such environments are often modeled as partially observable Markov decision processes (POMDPs) [55].

POMDPs can model a wide variety of control problems. In this paper, however, we focus on domains where an agent must execute actions to obtain a goal. Upon reaching the goal, the execution is stopped. Such POMDP problems are known as goal-POMDPs [13]. In a goal-

✉ Oded Blumenthal
odedblu@post.bgu.ac.il

Guy Shani
shanigu@bgu.ac.il

[1] Software and Information Systems Engineering, Ben Gurion University, Be'er Sheva, Israel

Published online: 08 July 2024

POMDP, there is a cost for every action, except at goal states. This provides an incentive for the agent to reach and remain at the goal. Often, reaching the goal requires a lengthy sequence of actions.

POMPD models allow us to reason about the hidden state of the system, typically using a belief state – a distribution over the possible environment states. The belief state can be updated given the executed action and the received observation. One can compute a policy, a mapping from beliefs to actions, that dictates which action should be executed given the current belief. Many algorithms were suggested for computing such policies [51].

However, in larger environments, it often becomes difficult to maintain a belief state, let alone compute a policy for all possible belief states. In such cases, one can use an online replanning approach, where after every action is executed, the agent computes which action to execute next. Such online approaches replace the lengthy single policy computation which is done offline, before the agent begins to act, with a sequence of shorter computations, which are executed online, during execution, after each action [46].

POMCP [54] is such an online replanning approach, extending the UCT algorithm [15, 36] for fully observable Markov decision processes (MDPs) to POMDPs. POMCP operates by constructing online a search tree, interleaving decision and observation nodes. Each decision node has an outgoing edge for every possible action, ending at an observation node. The outgoing edges from an observation node denote the possible observations that result from the incoming action. Each node is associated with a value representing the expected utility at that node.

The root of the tree is a decision node. To act, the agent chooses at the current decision node the outgoing action that leads to the highest value child. Then, the agent executes that action, and receives an observation. Using that observation the agent moves along the corresponding action and observation edges from the current decision node to the next decision node. To decide which action to take, the agent must hence compute a value, estimating the utility, for every child node. This is done by traversing the tree until leaf nodes have been reached.

To evaluate the value of leaf nodes, POMCP executes a random walk, known as a rollout, where the agent selects actions from some rollout policy to construct a trajectory and obtain an estimation of the quality of the leaf node. As a walk directly in belief space requires costly belief updates, POMCP instead samples a state from the leaf node, and executes the random walk in the fully observable state space. Following this estimation the values are propagated backwards towards the root decision node.

Clearly,the evaluation of the value at leaf nodes is highly dependant on the ability of the rollout policy to reach the agent goals. In complex problems, obtaining the goal may require a lengthy sequence of actions [37], and until the goal is reached, no meaningful information with respect to the goal is obtained. Indeed, practitioners that use POMCP often implement complex domain-specific heuristics for the rollout policy.

In this paper we focus on suggesting domain-independent heuristics for rollout policies. We leverage work in automated planning, using heuristics defined for classical and contingent planning problems [24, 26, 27]. POMCP methods often completely avoid a model representation, relying only on a simulator for executing actions and observing their outcomes. It is difficult, though, to compute a useful heuristic using only such a simulator, which is one reason that POMCP is typically limited to random walks. We thus represent POMDP problems in a structured manner, using boolean facts to capture the state of the environments. This allows both for a compact representation of large problems, compared with standard flat representations that do not scale, as well as the ability to use classical planning heuristics.

We begin by suggesting using the well-known h_{add} heuristic for choosing rollout actions [12]. This heuristic searches forward in a delete relaxation setting, until the goal has been

reached. Then, the value of an action is determined by the number of steps in the delete relaxation space following the action, required for obtaining the goal.

Next, we observe that any state-based rollout policy is inherently limited in its ability to evaluate the missing information required for reaching the goal, and hence, provide some estimate as to the value of information [31] of an action. We hence suggest a multi-state rollout policy, where actions are executed on a set of states jointly, and observations are used to eliminate states that are incompatible with the observed value. We show that this heuristic is much more informed in domains that require complex information-gathering strategies.

For an empirical evaluation, we extend domains from the contingent planning community with stochastic effects. We evaluate our heuristics, comparing them to random rollouts, showing that they allow us to provide significantly better behavior.

2 Background

We now provide the required background on MDPs, POMDPs, contingent planning, domain-independent heuristics, and the POMCP algorithm.

2.1 MDPs and POMDPs

A Markov decision process (MDP) [9, 43] is a decision making model, designed to take into account future decisions while deciding on the next action. We focus here on goal-based models, where an agent must reach a goal. Formally, a goal-MDP is a tuple $\langle S, s_0, A, tr, C, G\rangle$ where S is a set of states, s_0 is the initial state of the system, A is a set of actions, tr is a transition function, C is a cost function, and G is a set of goal states, where the execution terminates.

The state space S models all the relevant information for making a decision. The system is always at exactly one state, and this state is known to the decision maker. The action set A models the actions that are available to the decision maker. At each phase the decision maker observes the current state, and decides on an action. The system then transitions to a new state, and the process is repeated.

The tr function models stochastic state transitions, that is, $tr(s, a, s') = pr(s'|s, a)$ is the probability of transitioning to state s' given that action a was executed at state s. An MDP assumes the Markovian property, where only the current state influences the transition, i.e., $pr(s_{t+1}|s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}) = pr(s_{t+1}|s_{t-1}, a_{t-1})$.

The cost function $C : S, A \leftarrow R$ specifies a positive cost for every non-goal state. That is, for every $s \notin G$, $C(s) > 0$. The cost on goal states, however, is 0. The agent must minimize the expected sum of costs, and hence, an optimal policy would direct the agent to a goal state, and remain there.

The state space is often represented in a factored manner, using state variables. That is, a state s is composed as an assignment of values to a set of state variables $v_1, ..., v_k$. This can allow us to efficiently model the transition and cost functions.

A solution to a MDP can be represented as a *policy*, a function that assigns an action to every state. An optimal policy π^* minimizes the expected sum of costs — $\mathbb{E}[\sum_i C(s_i, \pi(s_i))]$, where s_0 is the initial state, and s_{i+1} is sampled from the $tr(s_i, \pi(s_i), \cdot)$ distribution.

A goal-oriented partially observable Markov decision process (goal-POMDP) is a tuple $\langle S, A, tr, C, G, \Omega, O\rangle$ [32]. S, A, tr, C, G form an MDP, often called the underlying MDP of the POMDP. Ω is the set of observations the agent can obtain. $O : S \times A \times \Omega \rightarrow [0, 1]$ is

the observation function, such that $O(s, a, o)$ is the probability of observing o when a was performed and led to state s. G is a set of goal states.

Because the state of a POMDP is partially observable, the agent typically is uncertain of the true underlying state of the world. Hence, it can maintain a *belief state* b, which is a distribution over S. That is, $b(s)$ is the likelihood that s is the current state based on the history of actions and observations that were observed by the agent. By probability laws, $0 < b(s) < 1$ for all $s \in S$ and $\sum_s b(s) = 1$. The cost function can then be defined over belief states: $C(b, a) = \sum_{s \in S} b(s) \cdot C(s, a)$.

In a goal-POMDP, the agent must ensure that the goal was reached in all possible states. That is, $sum_{s \in G} b(s) = 1$. Then, the execution terminates. The agent may be able to directly observe that the goal was reached, but this is not required. In many domains the agent may reason, from the sequence of actions and observations executed thus far, that it must be at a goal position. In many domains it may be impossible for such reasoning to occur, and the agent may never be completely certain that it has achieved the goal. Such domains are inappropriate for a goal-POMDP formalization (Fig. 1).

To maintain the current belief, after each action, the agent must re-compute its belief state and then decide on the next action given the new belief state. Given a belief state b, an action a, and an observation o, we can compute the next belief state b_a^o, which is reached when a is executed at b and o is observed:

$$b_a^o(s') = \frac{O(s', a, o) \cdot \sum_{s \in S} tr(s, a, s') \cdot b(s)}{Pr(o|a, b)} \tag{1}$$

2.2 Online planning

Offline POMDP solvers compute a policy, a mapping from beliefs to actions, before starting to act. Such policies specify which action should be taken in every potential belief state. Due to the lengthy policy constructing process, such approaches are typically only useful when working with small to medium-sized domains. An alternative is to implement an online strategy that only searches for local policies that are relevant to the agent's current belief state. The benefit of this method is that it only needs to consider belief states that are reachable from the current belief state [46].

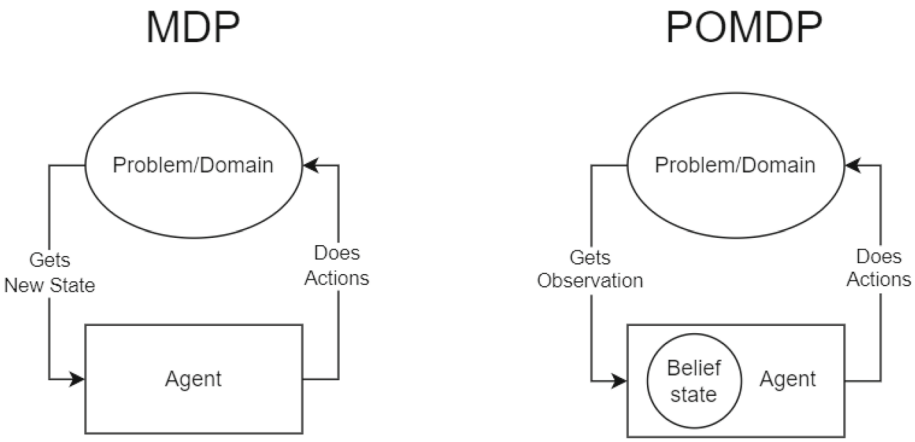


Fig. 1 MDP and POMDP visualization

Additionally, because online planning is performed at each step, it is sufficient to calculate only the maximal value for the current belief state, allowing for an anytime algorithm. That is, using a limited amount of time or iterations before being stopped and providing a policy for the current belief. A planning phase and an execution phase are applied alternately at each time step in an online algorithm. The current belief state of the agent is provided to the algorithm during the planning phase and determines the best course of action to take in that belief.

Usually, two steps are needed for doing this. By considering various potential sequences of actions and observations that might be taken from the current belief, a tree of belief states that can be reached from the current belief state is first constructed. This can be done using Bellman's equation, where the value of the current belief is calculated given values for its children:

$$V(b) = \min_{a \in A} C(b, a) + \sum_{o \in \Omega} pr(o|b, a) V(b_a^o) \tag{2}$$

We can hence propagate the value estimations up from the fringe nodes through their ancestors all the way to the root.

Once the values for all child nodes of the current decision nodes where estimated, we can choose the action that leads tot he child with the highest estimated value. We then execute it, receive an observation, and move to the next decision node. We also update the current belief state given the action and observation. We then run the value estimation phase again, act, and so forth, until the goal is reached.

2.2.1 Rollouts-based planning

In many cases, online approaches do not require an explicit model representation, using instead a black-box simulator, known as a generative engine. Such an engine accepts as input a state s and action a, and produces a new state s' that is sampled from the $tr(s, a, \cdot)$ distribution. In the partially observable case, the engine may also return an observation o, sampled from the $O(a, s', \cdot)$ distribution. This allows an online algorithm to avoid maintaining and reasoning about the exact belief, using a particle filter instead [54].

Rollout-Based Planning was presented by [36] as a method for planning through the simulation of episodes. The agent uses some policy, such as a random walk, to generate episodes. The agent then accumulates the costs of the simulated episodes to decide on the next action.

An episode is hence, in the fully observable case, a sequence of state-action-cost triplets. These can be generated using a generative engine, without an explicit model of the environment. As in the online methods above, a look-ahead tree is constructed by incrementally adding the knowledge collected during a single episode to the tree via a rollout-based algorithm.

We can maintain track of estimates of the action values at states that were already encountered in prior episodes. As a result, if a certain state is visited again, the previous collected estimates can be reused, thereby accelerating the convergence of the value estimations.

Such a rollout approach hence iteratively produces episodes that estimate the value of possible actions. Then, it selects the action with the highest average observed long-term utility. The strategy that selects and performs actions until a terminal condition is satisfied is called the rollout policy. The terminal condition could be the depth at which an episode is cut, or reaching a goal state.

In the basic vanilla Monte Carlo Planning [21], the actions are chosen during the episodes uniformly. However, the efficiency of the whole algorithm is heavily influenced by the ability of the policy to decide on good actions that lead towards the goal. Rollout-based algorithms that employ selective sampling may be superior to other techniques in domains where the set of successor states is relatively small [36].

2.2.2 POMCP

The POMCP (Partially Observable Monte-Carlo Planning) algorithm is an online replanning method for acting in POMDPs. POMCP uses a Monte Carlo tree search (MCTS) approach to select the next action to execute [54]. At each replanning episode POMCP constructs a tree, where the root node is the current belief state. Then, POMCP runs forward simulations, where a state is sampled from the current belief, and actions are chosen using an exploration strategy that initially selects actions that were not executed a sufficient amount of times, but gradually moves to select the seemingly best action. Observations in the simulations are selected based on the current state-action observation distribution.

When reaching a leaf node, POMCP begins a rollout. This rollout is designed to provide a value estimate for the leaf, based on some predefined rollout policy. The value of the leaf is updated using the outcome of the rollout, and then the values of the nodes along the branch that were visited during the simulation are updated given their descendants. Obviously, the values of all nodes in the tree are hence highly dependent on the values obtained by the rollout policy.

POMCP is an anytime algorithm, continuing to run simulations until a timeout, and then returning the action that seems best at the root of the search tree.

POMCP was designed for large problems. Therefore, POMCP does not maintain and update a belief state explicitly. Instead, POMCP uses a particle filter approach, where a set of states is sampled at the initial belief, and this set progresses through the tree.

2.3 Contingent planning under partial observability

A partially observable contingent planning problem is a tuple: $\pi = \langle P, A_{act}, A_{sense}, \varphi_I, G \rangle$ [5, 14, 29, 49]. P is a set of facts, A_{act} is a set of actuation actions, and A_{sense} is a set of sensing actions. φ_I is a formula describing the set of initially possible states. For ease of exposition, we will assume that φ_I is a conjunction of facts, disjunctions of facts, or *oneof* clauses over facts, specifying that exactly one fact in the clause holds. A state s assigns truth values to all $p \in P$. G is a formula over P defining goal conditions.

An actuation action $a \in A_{act}$ is a pair, $\{pre(a), \mathit{eff}(a)\}$, where $pre(a)$ is a set of fact preconditions, and $\mathit{eff}(a)$ is a set of pairs (c, e) denoting conditional effects. We use $a(s)$ to denote the state that is obtained when a is executed in state s. A sensing action $a \in A_{sense}$ is a pair, $\{pre(a), obs(a)\}$, where $pre(a)$ is as above, and $obs(a)$ is a set of facts in P whose value is observed when a is executed. We denote by $obs(a, s)$ the values of the observed facts when a is executed at state s. This separation to actuation and sensing actions is only for ease of exposition, and our methods apply also to actions that both modify the state of the world and provide an observation.

In contingent planning, a belief-state is the set of possible states. This is different than the stochastic POMDP belief, which is a distribution over possible states. In a non-stochastic contingent planning setting, the belief can used to decide whether the preconditions of an

action hold for all current states, and whether the goal has been reached for all possible states. This non-stochastic notion of a belief is also useful in our stochastic scenario, for these two tasks. The initial belief state, $b_I = \{s : s \models \varphi_I\}$ is the set of initially possible states.

Preconditions allow us to restrict our attention only to applicable actions. An action is applicable in a belief b is all possible states in b satisfy the preconditions of the action. Obviously, one can avoid specifying preconditions for actions, allowing for actions that can always be executed, as is typically the case in flat POMDP representations. However, in domains with many actions, preconditions are a useful tool for drastically limiting the amount of actions that should be considered.

2.3.1 Regression-based belief maintenance

The non-stochastic belief can be represented by a set of states. However, for large domains, these states can often be more efficiently represented by a set of boolean formulas. When executing an action and sensing an observation, these formulas can be updated to represent the new belief state. However, in many cases, these formulas rapidly become very complicated, representing complex relationships between the state variables.

Thus, updating a non-stochastic belief can be costly. Alternatively, one can avoid the computation of new formulas representing the updated belief, by maintaining only the initial belief formula, and the history — the sequence of executed actions and sensed observations [18].

When regressing a literal l through an action a, we check which conditions must hold prior to the execution of a, so that l will hold following the action. This can be used in two ways. First, we can regress the negation of the precondition of an action. If the regressed formula is inconsistent with the initial belief, then the preconditions must hold. Similarly, we can also regress the negation of the goal, to see whether the goal holds. Second, we can regress a received observation. The formula generated by such a regression constrains the initial belief, eliminating states that are inconsistent with the observed value. This helps us to reduce uncertainty following an observation.

When the agent needs to query whether the preconditions of an action or the goal hold at the current node, the formula is regressed [45] through the action-observation sequence back towards the initial belief. Then, one can apply SAT queries to check whether the query formula holds. This approach can be highly useful for larger POMDPs, complementing the particle filter approach used in POMCP.

We now briefly review the regression mechanism that we use. First, let us consider the regression of an actuation action a that does not provide an observation. Let ϕ be a propositional formula and a a *deterministic* actuation action. Let $c_{a,l}$ denote the condition under which literal l is an effect of a, and that $a(s)$ satisfies l iff either $s \models c_{a,l}$ or $s \models l \wedge \neg c_{a,\neg l}$. Hence, we define the *regression* of ϕ with respect to a as:

$$rg_a(\phi) = pre(a) \wedge \phi_{r(a)} \tag{3}$$

$$\phi_{r(a)} = \text{replace each literal } l \text{ in } \phi \text{ by } c_{a,l} \vee (l \wedge \neg c_{a,\neg l}) \tag{4}$$

Now, consider a sensing action and an ensuing observation. Suppose we want to validate that ϕ holds following the execution of $a \in A_{sense}$ in some state s given that we observed

$obs(a) = o$. Thus, we need to ensure that following a, if l holds then ϕ holds. That is:

$$rg_{a,o}(\phi) = rg_a(obs(a) = o \rightarrow \phi) \tag{5}$$

Regression maintains the equivalence of the formula [45]. For any two formulas ϕ_1 and ϕ_2 we have:

1. $\phi_1 \equiv \phi_2 \Rightarrow rg_{a,o}(\phi_1) \equiv rg_{a,o}(\phi_2)$
2. $\phi_1 \equiv \phi_2 \Rightarrow rg_a(\phi_1) \equiv rg_a(\phi_2)$

Hence, we can produce a regression over formulas, and compare the regressed formulas, making conclusions about the original formulas.

The regression can be recursively applied to a sequence of actions and observations (history) h as follows:

$$rg_{h+(a,o)}(\phi) = rg_h(rg_{a,o}(\phi)); \quad rg_{\epsilon,\epsilon}(\phi) = \phi \tag{6}$$

where ϵ is the empty sequence. This allows us to perform a regression of a formula through an entire plan, and analyze the required conditions for a plan to apply.

In addition, the regression mechanism of Brafman and Shani [18] maintains a cached list of fluents $F(n)$ that are known to hold at node n, given the action effects or observations. Following an actuation action a, $F(a(n))$ contains all fluents in $F(n)$ that were not modified by a, as well as effects of a that are not conditioned on hidden fluents. For a sensing action revealing the value l, $F(a(n, l)) = F(n) \cup \{l\}$. During future regression queries, when a value at a particular node becomes known, e.g. when regressing a later observation, it is added to $F(n)$. All fluents p such that $p \notin F(n) \wedge \neg p \notin F(n)$ are said to be hidden at n. The cached list is useful for simplifying future regressed formulas.

3 Related work

Augmenting MCTS methods with heuristics in the context of fully observable MDPs was previously suggested. Keller and Helmert [34] describe an MCTS tree-based approach that uses planning-based heuristics. The PROST planner [33] uses a heuristic for estimating the value of states. The DP-UCT approach [52] uses a planning heuristic based on deep learning for the rollout phase. These approaches, focusing on MDPs, do not evaluate the heuristic value of a belief state, as we do. We are not aware of previous attempts to adapt these approaches to POMDPs.

There were several extensions suggested for POMCP [37]. For example Kurniawati and Yadav [38] consider dynamic environments, and Hörger, Kurniawati, and Elfes [30] consider non-linear dynamics. All these methods rely on rollouts, and can hence leverage the rollout strategies that we suggest here.

In Sunberg and Kochenderfer [60], there are multiple types of POMCP extensions. The first one is called POMCP with double progressive widening (POMCP-DPW), which handles domains with a large number of observations. With some probability, they discard observations during the simulation phase, and by that maintain a smaller observation branching factor. Another POMCP extension that is presented in Sunberg and Kochenderfer [60] is partially observable Monte Carlo planning with observation widening (POMCPOW). This algorithm uses re-samples through the simulation phase. POMCPOW has the advantage of generality per observation node in that manner, but also has data loss through decisions with a low number of simulations. Such techniques may also benefit from our domain independent heuristics, and can also apply to the symbolic representation that we use.

Sunberg and Kochenderfer [60] also suggest particle filter trees with double progressive widening (PFT-DPW), which uses the approach of Monte-Carlo Tree Search (MCTS) but instead of states, use belief particles. In each node, b' is sampled and progressed through the constructed MCTS tree. They also use belief-based rollouts, but only random rollouts without a heuristic. Our heuristics may hence improve on these techniques, providing better estimations with less rollouts.

Sunberg and Kochenderfer [60] also extend POMCP to the challenge of solving POMDPs with continuous state, action, and observation spaces. It also requires a rollout policy to evaluate the utility of leaf nodes. Our rollout strategies rely on classical planning approaches which are discrete, and it is hence unlikely that our methods can be easily extended to continuous domains.

DESPOT [57] is an online POMDP solver based on tree search, similar to POMCP. DESPOT uses a different strategy than the UCB rule for constructing the tree, designed to avoid the overly greedy nature of POMCP exploration strategy. DESPOT also requires a policy to evaluate the utility of a leaf in the tree, and the authors stress the importance of a strong default policy to improve convergence. Thus, our methods can be directly applied to DESPOT as well.

DESPOT summarizes the execution of all policies under so-called K-sampled scenarios. A K-sampled scenario is structurally similar to a standard belief tree but contains only belief nodes reachable under K simulations. This leads to a dramatic improvement in computational efficiency when K is small [57]. In our method, we also use sub-sampling from the belief state by particle filter, but in different from DESPOT we are using a larger K value(e.g 500).

An extension of DESPOT is the Regularized DESPOT (R-DESPOT). R-DESPOT interprets a lower bound as a regularized utility function, which it uses to optimally balance the size of a policy and its estimated performance under the sampled scenarios [57]. In contrast to R-DESPOT, we do not dynamically change the size of the policy, and it is set to a constant value.

In addition Luo, Bai, Hsu, and Lee [40] extend DESPOT, and create an algorithm called importance sampling DESPOT (IS-DESPOT). In order to improve the performance in cases of uncertainty. By using importance sampling on DESPOT, performance on critical but rare events, which are difficult to sample, increases. In our method, we did not give special treatment for uncommon events, and let their probability to occur determine if they are important enough to be considered.

Cai, Luo, Hsu, and Lee [19] present a method called Hybrid Parallel DESPOT (HyP-DESPOT), which combines CPU and GPU to accomplish real-time online planning for domains with significant state, action, and observation spaces. HyP-DESPOT utilizes multi-core CPUs to conduct parallel DESPOT tree searches by simultaneously evaluating a number of different independent paths. For the purpose of running concurrent Monte Carlo simulations at the search tree's leaf nodes, GPUs are used. It is likely that our methods can also benefit from multi-core and GPU architectures.

Saborío and Hertzberg [47] suggest a method called PSG to evaluate the proximity of states to the goal. They suggest using PSG in several places within POMCP including rollouts. PSG assumes that states are defined in a factored manner using state features, and computes a function from features to the goal using subgoals. In essence, their approach can be considered as a type of heuristic, which is highly related to the concept of landmarks in classical planning [44]. Representing the POMDP as a stochastic contingent planning problem, as we do, allows us to use any heuristic developed in the planning community, and can hence be considered to be an extension of PSG.

In Saborío and Hertzberg [48], the researchers used a method called incremental refinement (IRE) to give each state-action pair a relevance score. With those relevance scores, they decided which branches were less meaningful and decreased the average branching factor of the constructed tree. Our algorithm does not apply any type of pruning, but it's less needed because in POMCP we mainly focus on the high-value branches in most of the simulations.

Similarly, Xiao, Katt, ten Pas, Chen, and Amato [62] also find it difficult to provide good rollout strategies to compute the value of POMCP leaves. Focusing on a robotics motion planning domain, they suggest SVE, a state value estimator, that attempts to evaluate the utility of a state directly.

Saboŕo and Hertzberg [35] also focus on the need to use heuristics for guiding search in POMDPs. They focus on RTDP-BEL [14], an algorithm that runs forward trajectories in belief space to produce a policy. They show that using a heuristic can significantly improve RTDP-BEL. They use domain-specific heuristics, and as such, our domain independent approach can also be applied to their methods.

In the field of domain-independent heuristics, [53] and [61] suggest machine learning methods for creating domain-independent heuristics by learning solutions of a domain and applying them to other problems of that domain. We, on the other hand, do not need a training set of solutions for every domain we try to solve and furthermore, we do not require initialization time to learn the domain's heuristic.

An example of a domain-independent heuristic is presented in Haslum et al. [25]. This paper focuses on the Pattern Database (PDB) heuristic, with an extension to construct the patterns in an automatic way, depending on the search space size they are able to make PDB domain-independent. This matches the intuitions in our suggested method and shows the importance of good domain-independent heuristics in the field of planning and in specific progressive (forward-search) planning.

Bertoli, Cimatti, Roveri, and Traverso [11], construct an algorithm based on And-Or trees that contains belief states in each node, for strong planning. The article defines strong planning as planning that considers the entire belief state when executing an action, and in simple words, the agent cannot take an action whose preconditions are not satisfied across all states in the belief state. This is similar to the intuition behind our belief-based heuristic.

4 POMCP for stochastic contingent planning

We now describe the adaptation of POMCP to goal-oriented POMDP domains specified as stochastic contingent planning problems. We now define this concept formally.

Boolean formulas are used in contingent planning for efficiently describing preconditions, effects, goal conditions, and the initial belief. To achieve this efficient representation for a stochastic domain, we must hence extend the concept of boolean formula used in contingent planning to the stochastic case. We define a stochastic formula ψ to be a set of options, representing a set of disjoint alternatives that may hold, and their probabilities. One can think of such a formula as a stochastic version of a *oneof* statement. We restrict each option o to be a conjunction of facts, associated with a probability $pr(o) \in (0, 1)$ such that $\sum_o pr(o) = 1$. One can sample a single option from the stochastic formula, given the distribution defined by $pr(o)$.

A stochastic contingent planning problem is a tuple $\pi = \langle P, A_{act}, A_{sense}, \varphi_I, pr_I, G \rangle$, where $P, A_{sense}, \varphi_I, G$ are as in a deterministic contingent planning problem. pr_I is a stochastic formula defining probabilities over the initial values of some unknown facts. For

each action $a \in A_{act}$, the formula defining the effects of a may contain stochastic formulas, capturing stochastic effects. We denote by $a(s)$ the distribution over next states given that a was executed at s.

This definition does not support noisy observations, however, this is not truly a limitation. One can compile a noisy observation into a deterministic observation over an artificial fact whose value changes stochastically. Consider, for example, a sensor that noisily detects whether there is a wall in front of a robot. Instead of noisily observing whether there is a wall, we can deterministically detect a green light that is lit when the sensor (stochastically) detects a wall. That is, we can observe the green light without noise, but the green light is only noisily correlated with the existence of a wall.

Algorithm 1 POMCP for stochastic contingent problems.

Search(h)
  while *timeout not reached* **do**
    $s \sim \varphi_I$, pr_I;
    $s' \leftarrow$ apply h to s;
    $Simulate(s', root, o)$
Simulate($s, n, depth$)
  add s to $n.b$;
  $count(n) \leftarrow count(n) + 1$;
  if G *is satisfied in* $n.history$ **then**
    $V(n) \leftarrow 0$;
    return
  if $depth > MaxTreeDepth$ **then**
    $V(n) += Rollout(s, n.b)$;
    return
  if n *is a leaf node* **then**
    for $a \in A_{act} \cup A_{sense}$ **do**
      if $pre(a)$ *are satisfied at* $n.history$ **then**
        Add child $n.a$ to n;
        for $o \in obs(a)$ **do**
          Add child $n.a.o$ to $n.a$;
  $a \leftarrow argmin_a Q(n, a) - c\sqrt{\frac{log(count(n))}{count(n.a)}}$;
  if $a \in A_{act}$ **then**
    $s' \sim a(s), o \leftarrow null$
  else
    $s' \leftarrow s, o \leftarrow obs(a, s)$
  $Simulate(s', n.a.o, depth + 1)$;
  $count(n.a) \leftarrow count(n.a) + 1, count(n.a.o) \leftarrow count(n.a.o) + 1$;
  $V(n.a) \leftarrow \frac{\sum count(n.a.o) \cdot V(n.a.o)}{count(n.a)}$;
  $V(n) \leftarrow \min_a V(n.a)$
Rollout(s, B)
  $depth \leftarrow 0$;
  while $s \notin G \wedge depth < MaxRolloutDepth$ **do**
    $a \leftarrow \pi_{rollout,B}(s)$;
    $s \sim a(s)$;
    if a *is a sensing action* **then**
      Remove from B states that do not agree with s on the observation
    $depth \leftarrow depth + 1$
  return $depth$

Algorithm 1 describes the POMCP implementation for stochastic contingent planning problems. When the agent needs to act, it calls Search. Search repeatedly samples a state (line 3-4) and simulates forward execution given this state is the true underlying system state.

We do not maintain or update a belief state. Instead, we use regression over the history of executed actions and sensed observations. We use regression for two specific tasks. First, before executing an action we must ensure that its preconditions hold in the current belief. That is, that for each state s s.t. $b(s) > 0$, $s \models pre(a)$. We maintain during planning an inexact probabilistic estimation of the belief, through a particle filter. It may hence happen that although for some state s, the true probability $b(s) > 0$, while in the particle filter approximation s is not represented.

The second task is when ensuring the the current belief satisfies the goal. That is, that for every state s s.t. $b(s) > 0$, $s \models G$. Again, due to the inexact particle filter, we cannot rely only on the states in the current particles to satisfy the goal. Regression hence allows us to ensure, in both cases, that although the particle filter is inexact, all actions are soundly executed, and when we stop, the goal was achieved for all possible states.

The Search procedure hence samples a state s from the initial belief state, given the initial probability distribution pr_I (line 3). Then, the agent advances the sampled state through the history to obtain a current state s' (line 4).

The Simulate procedure is recursive. We first check whether the current tree node is a goal belief. This is done be regressing the negation of the goal formula $\neg G$ through the history of the current node. For goal beliefs, the value is 0, and we can stop.

Our implementation of POMCP also stops deepening the tree after a predefined threshold Max-Tree-Depth. If that threshold is reached (line 12), we run a Rollout to compute an estimation for the cost of reaching the goal from this node (line 13).

In line 15 we check whether this node has already been expanded, and if not, we compute its children. We do so only for applicable actions whose preconditions are satisfied in the current belief (line 17). Again, this is computed using regression over the history.

We now select an action a using the UCT exploration-exploitation criterion (line 21), and sample a next state and an observation (lines 22-27). We call Simulate recursively in line 28.

Lines 29-32 update the value of the current node. Lines 28,29 update the counters for the executed action and received observation. Line 31 computes the value for the action as a weighted average over all observations. Line 32 computes the value of the node as the minimal cost among all actions. Our value update, which we empirically found to be more useful, is different than the original POMCP, which uses incremental updates, and more similar to the value update in DESPOT [40].

The Rollout procedure receives as input the current simulated state s, as well as a set B of states (particles) in the node from which the rollout begins. B is used by some of our rollout heuristics, as we explain below. The rollout executes actions given the heuristic rollout policy until the goal has been reached, or a maximal number of steps has been reached.

5 Domain independent heuristics for POMCP

We now describe the main contribution of this paper — two domain independent rollout heuristics that leverage methods developed in the automated planning community, using the structure specified in the stochastic contingent planning problem.

5.1 Delete relaxation heuristics

Delete relaxation heuristics are built upon the notion that if actions have only positive effects, then the number of actions that can be executed before the state becomes fixed is finite, and in many cases, small. Also, as actions cannot destroy the precondition of other actions, one can execute actions in parallel. Algorithm 2 portrays a delete relaxation heuristic.

Delete relaxation heuristics create a layered graph, interleaving action and fact layers. The first layer, which is a fact layer, contains all the facts that hold in the state for which the heuristic is computed (line 2). The second layer, which is an action layer, contains all the actions whose preconditions hold given the facts in the first layer (line 5). The next layer, which is again a fact layer, contains all the positive effects of the actions in the previous layer, as well as all facts from the previous layer (line 6), and so forth. We stop developing the graph once no new facts can be obtained (line 8).

After the graph is created, one can compute a number of heuristic estimates. The h_{max} returns the depth of the first fact layer that satisfies G. The h_{add} heuristic sums the fact depth of all goal facts (line 9) [12]. The h_{ff} heuristic computes a plan in the relaxed space by tracing back actions that achieved the goal predicates [28].

In this paper we experimented using the h_{add} heuristic.

Algorithm 2 Single state h_{add}.

1 $\pi_{h_{add}}(s)$
2 $\quad fact_0 \leftarrow$ all facts in s;
3 $\quad i \leftarrow 1$;
4 $\quad$ **repeat**
5 $\quad\quad action_i \leftarrow \{a \in A_{act} : fact_{i-1} \models pre(a), a \notin \bigcup_{j=1..i-1} action_j\}$;
6 $\quad\quad fact_i \leftarrow fact_{i-1} \cup \{f \in eff(a) : a \in action_i\}$;
7 $\quad\quad i \leftarrow i + 1$;
8 $\quad$ **until** $fact_{i-1} = fact_{i-2}$;
9 $\quad$ **return** $\sum_{f \in G} i : f \in fact_i, f \notin fact_{i-1}$

5.2 Heuristics in belief space

A major disadvantage of the above heuristics is that they focus on a single state. When the agent is aware of the true state of the system, observations have no value. Hence, the above heuristics, as well as any heuristic that is based on a single state, do not provide an estimate for the value of information, which is a key advantage of POMDPs. We hence suggest now a heuristic that is computed over a set B of possible states (Algorithm 3).

We compute again the delete relaxation graph, with a few modifications. We compute for each state in B a separate fact layer, where $fact_i^s$ is the fact for state s at level i in the graph. This is similar in spirit to maintaining $p|s$ in translations from contingent to classical planning [17].

An action can be applied only if its preconditions are satisfied in the fact layers of all agents (line 7). This is equivalent to the requirement in contingent planning where an action is applicable only if it is applicable in all states in the current belief, where B is served as an approximation of the true belief state.

Second, our method leverages the deterministic observations, that allow us to filter out states that are inconsistent with the received observation (lines 8-14). When a sensing action can be applied, all states that do not agree with the value of s on the observation are discarded from B (lines 10-14). That is, we remove the fact layers corresponding to these states, and no longer consider them when computing which actions can be applied.

Algorithm 3 Belief space h_{add}.

```
1  π_{h_add}(s, B)
2      ∀s' ∈ B, fact_0^{s'} ← all facts in s';
3      B_0 ← B;
4      i ← 1;
5      repeat
6          B_i ← B_{i−1};
7          action_i ← {a ∈ A_act : ∀s' ∈ B_i, fact_{i−1}^{s'} ⊨ pre(a), a ∉ ⋃_{j=1..i−1} action_j};
8          for a ∈ A_sense do
9              if ∀s' ∈ B_i, fact_{i−1}^{s'} ⊨ pre(a) then
10                 F ← the values of obs(a) in fact_i^s;
11                 for s' ∈ B_i, s' ≠ s do
12                     F' ← the values of obs(a) in fact_i^{s'};
13                     if F' ≠ F then
14                         B_i ← B_i \ {s'};
15         ∀s' ∈ B_i, fact_i^{s'} ← fact_{i−1}^{s'} ∪ {f ∈ eff(a) : a ∈ action_i};
16         i ← i + 1;
17     until B_{i−1} = B_{i−2} ∧ ∀s' ∈ B_{i−1} : fact_{i−1}^{s'} = fact_{i−2}^{s'};
18     return ∑_{f∈G} i : f ∈ fact_i^s, f ∉ fact_{i−1}^s
```

We stop when both no states were discarded, and no new facts were obtained (line 17). This process must take into account sensing actions to remove states that are incompatible with s, which would allow, at the next iteration, that action preconditions would be satisfied for less states, and hence additional actions can be executed.

6 Empirical evaluation

We conduct an empirical study to evaluate our methods. Our methods are implemented in C#.

6.1 Benchmark domains

We extended the following contingent planning benchmarks to stochastic settings:

Doors In the door domain the agent must move in a grid to reach a target position. Odd levels in the grid are all open, while in even levels there are doors, and only one door is open. The agent can sense whether a door is open when it is at adjacent cells. The agent must identify the open doors and get to the target position. In the stochastic version the agent can open a closed door with some probability of success. The agent can hence either search for the already open door, or attempt to open a closed door.

Blocks world In the contingent blocks world problem, the agent does not know the structure of the initial block configuration, but it can sense whether one block is on top of another one, and whether a block is clear. In the stochastic version moving a block from one block to another has a 0.3 probability of success, while moving blocks to and from the table succeeds deterministicly. Hence, it is often preferable to use the table as an intermediate position.

Unix In this domain the agent must search for a file in a file system, and copy it to a destination folder. In the stochastic version there is a non-uniform distribution over the possible locations of the file.

MedPks The agent here needs to identify which illness a patient has and treat it. To do so, the agent tests for each illness independently, until the proper illness is found. The stochastic version here has non-uniform distribution over the possible illnesses as well.

Localize In this domain the agent must reach a goal position in a grid. The agent does not know where it initially is, and can only sense adjacent walls. In the stochastic version there several places in the grid where the agent may slip and stay in place. This makes the localization in the grid more difficult.

Wumpus In this challenging problem the agent must reach a target position in a grid infested by monsters called Wumpuses. Cells may be unsafe to travel as they may contain either a Wumpus or a pit. Wumpuses emit a stench, and pits emit a breeze, both of which can be sensed in adjacent cells. The agent must sense in multiple positions to identify the safe cells. The stochastic version here also has non-uniform distribution over the safe cells.

6.2 Results

For each problem above we run 20 online episodes, and compute the success rate, the average run time for computing the next action, and the average cost to the goal. We did not use a timeout, but runs longer than 100 steps were considered to be stuck in a loop, and terminated.

Tables 1, 2, and 3 present the experiments results over the benchmarks, comparing the random (uniform) rollout policy (denoted Rnd), the h_{add} heuristic using a single state ($\pi_{h_{add}}(s)$), and the h_{add} heuristic over multiple states ($\pi_{h_{add}}(s, B)$).

We begin by looking at the quality of the policy — the average cost to the goal. As can be seen in Table 1, the random rollout policy is best only in the MedPks domain, and close to best in unix. This is not too surprising, because in these two domains the best strategy is very simple, and random strategies easily stumble upon the goal. In blocks all methods achieved similar performance, because the optimal strategy is very short, and rollouts are less important. This domain has many possible actions, and hence a huge branching factor, making it difficult to scale up using POMCP.

With respect to success rates (Table 2)), on doors and localize, which require lengthier trajectories to reach the goal, but do not need long information-gathering efforts, the single

Table 1 Comparing rollout heuristics on various domains over 20 runs on each problem

Domain	Sim.	Avg cost		
		Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s, B)$
Doors 5	1500	17.3	**13.7**	14.4
Blocks 4	1500	4.9	4.45	4.8
Localize 3	1500	14.75	**10.35**	**10.95**
MedPks 10	1500	**6.3**	7.25	7.45
Unix 1	1500	7.35	**5.65**	6.4
Wumpus 5	1500	39.47	33.352	**24.33**
Wumpus 5	500	56.117	33.722	**22.05**

For each domain, we compute the average cost of plans generated for each of the tested heuristics
Bolded entries were found significant using a two-tailed T test

Table 2 Comparing rollout heuristics on various domains over 20 runs on each problem

Domain	Sim.	Success rate		
		Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s, B)$
Doors 5	1500	100%	100%	100%
Blocks 4	1500	100%	100%	100%
Localize 3	1500	80%	**100%**	**100%**
MedPks 10	1500	100%	100%	100%
Unix 1	1500	100%	100%	100%
Wumpus 5	1500	85%	85%	**90%**
Wumpus 5	500	85%	90%	**100%**

For each domain, we compute the portion of runs where the agent successfully reached the goal for each of the tested heuristics
Bolded entries were found significant using a two-tailed T test

state h_{add} strategy operates very well. However, on Wumpus, where relatively long sequences of actions are needed for information gathering, the multiple-state heuristic works best, because it can capture the value-of-information provided by sensing actions. Admittedly, we expected more significant difference between the "smart" heuristics and the random rollout. Perhaps additional domains where value-of-information is important can provide stronger evidence of the importance of belief-based heuristics.

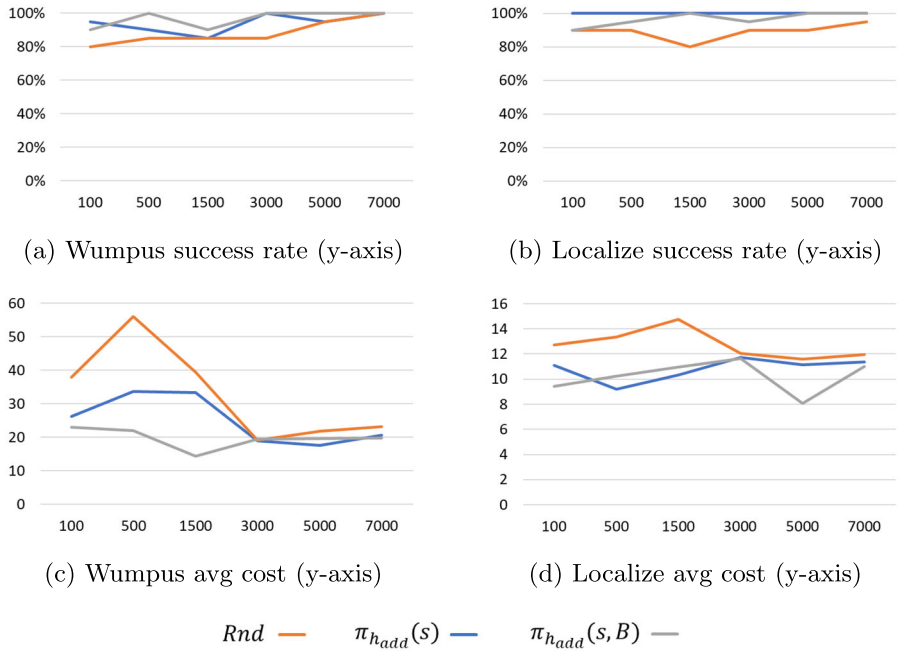
With respect to the required time to run the simulations for a single decision (Table 3), the results are mixed. While obviously the random strategy requires no time to compute the next action during a rollout, it often results in lengthy rollouts, which reduce this effect. The single state heuristic is almost always faster than the multi-state heuristic, but often not by much.

To farther understand the behavior of POMCP when varying the amount of effort invested in rollouts, we conducted further experiments. In one experiment, we vary the amount of rollouts before making a decision. In the second, we vary the amount of time dedicated to rollouts before making a decision. Time is not directly correlated to the amount of rollouts, because some rollout policies require more time for each rollout, and the quality of the policy also dictates shorter or longer rollouts. For example, on Wumpus, for 7000 rollouts, the random policy required 548 seconds (23.8 seconds per decision), the $\pi_{h_{add}}(s)$ policy required 84 seconds (4 seconds per decision), and the $\pi_{h_{add}}(s, B)$ policy required 261 seconds (13.17

Table 3 Comparing rollout heuristics on various domains over 20 runs on each problem

Domain	Sim.	Avg step time (secs)		
		Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s, B)$
Doors 5	1500	6.552	**0.403**	1.86
Blocks 4	1500	0.33	**0.13**	0.37
Localize 3	1500	1.325	**0.752**	1.16
MedPks 10	1500	4.029	4.594	3.388
Unix 1	1500	2.551	**0.274**	1.322
Wumpus 5	1500	10.829	**3.491**	4.909
Wumpus 5	500	2.442	**0.823**	1.12

For each domain, we compute the average time of the algorithm to make a step for each of the tested heuristics
Bolded entries were found significant using a two-tailed T test



(a) Wumpus success rate (y-axis)

(b) Localize success rate (y-axis)

(c) Wumpus avg cost (y-axis)

(d) Localize avg cost (y-axis)

Fig. 2 The effect of varying the number of rollouts (x-axis) on success rate and average cost for reaching the goal

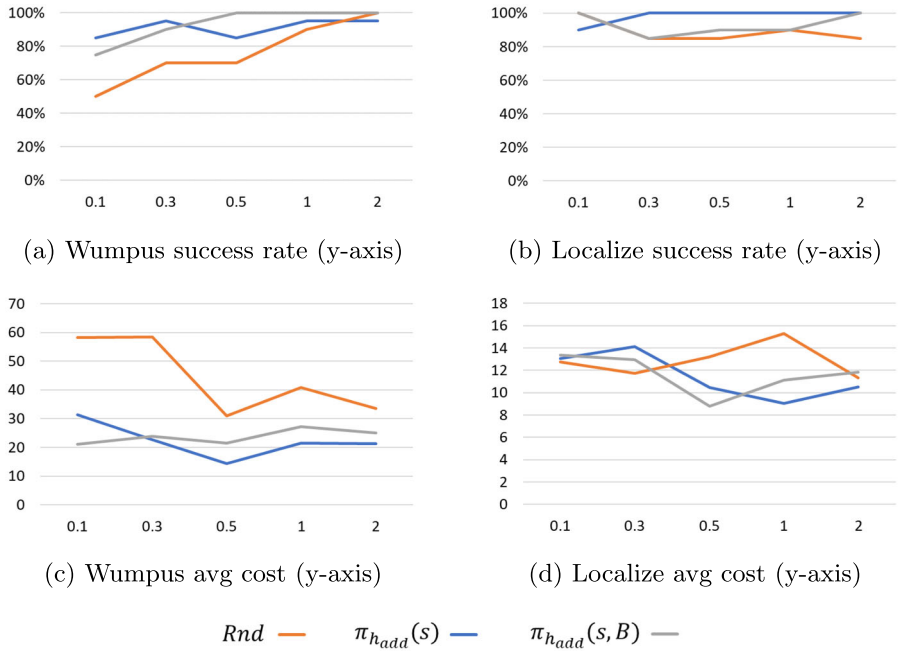
seconds per decision). As with the experiments above, we ran each configuration 20 times, and report averages.

Figure 2 shows the effect of varying the amount of rollouts on the quality of the POMDP policy. In Wumpus (Fig. 2a), it is typically the case that the rollout policies require a substantial amount of rollouts before reaching a success rate of 100%. We can see here that the $\pi_{h_{add}}(s, B)$ is better, consistently succeeding to reach the goal after 3000 rollouts, while random is, as expected, worst, requiring 7000 rollouts before achieving a success rate of 100%.

On Localize (Fig. 2b), the results are different. Random never reached a complete success rate, while $\pi_{h_{add}}(s)$ always consistently achieves a 100% success rate. Recall that on this version of Localize, there are two paths for reaching the goal, and the shorter path with respect to the number of steps has a higher cost, and should be avoided.

We now look at the effect of the amount of rollouts on the average cost for reaching the goal (computed only for executions that reached the goal) in Localize (Fig. 2d). We see that $\pi_{h_{add}}(s)$ did find a good policy with only 500 rollouts, but additional rollouts confused it, making the policy worse. We a similar problem with $\pi_{h_{add}}(s, B)$ with 5000 rollouts, which produces the best behavior here, and then a worse policy at 7000 rollouts. This phenomena often occurs in reinforcement learning, where an early policy works well, but additional training diverts the policy to worse behavior. It is likely that with many more rollouts the good behavior would return. Here, the random policy, with any amount of rollouts, did not discover the best path to the goal.

For Wumpus (Fig. 2c) the average cost is misleading, because we cannot comapre costs when the success rate is less than 100%. In such cases, a policy ay reach the goal only for easier states, and hence result in a lower average cost, and more failures. The behavior of



(a) Wumpus success rate (y-axis)

(b) Localize success rate (y-axis)

(c) Wumpus avg cost (y-axis)

(d) Localize avg cost (y-axis)

Fig. 3 The effect of varying the amount of time in seconds (x-axis) for rollouts on success rate and average cost for reaching the goal

the $\pi_{h_{add}}(s, B)$ policy with 500 and 1500 rollouts is an example of that. With 500 rollouts it reaches 100% success rate, with an average cost of 22.05. With 1500 rollouts the success rate drops to 90%, but it solves the easier cases very well, with an average cost of 14.33.

At 7000 rollouts, where all methods reach a complete success rate on Wumpus, $\pi_{h_{add}}(s)$ and $\pi_{h_{add}}(s, B)$ are not significantly different, while random is significantly worse.

Figure 3 analyzes the effect of limiting the runtime for every rollout session, i.e., the planning time per decision step, instead of the amount of rollouts. Arguably, for agents that operate in the real world, thresholding the wall clock time before making a decision is perhaps more reasonable in many applications, where the agent interacts with external events, generated by the environment, by other agents, or by humans. Consider for example a dialog agent that interacts with people. It is very reasonable to limit the amount of time that the person waits for the agent to respnd. As the amount of rollouts is not directly correlated with time, a threshold on the allowed time seems more reasonable.

The effect of increasing the amount of time is similar to increasing the amount of rollouts. In Wumpus we see an almost monotonic increase in success rate (Fig. 3a), but not in policy quality (Fig. 3c). On Wumpus $\pi_{h_{add}}(s, B)$ is better than $\pi_{h_{add}}(s)$, given the limited time. For Localize we see that random does not converge even after a long time. Here, $\pi_{h_{add}}(s)$ is better than $\pi_{h_{add}}(s, B)$ given limited time.

To summarize our findings, in problems that require an analysis of the value-of-information, our multiple-state heuristic is of value. In other domains, a state based domain independent heuristic works well. In any case, our domain independent heuristics are much better than the random rollouts that are often used in MCTS planning algorithms.

7 Conclusion

In this paper we suggested to model goal POMDPs as stochastic contingent planning models, which allows us to use domain independent heuristics developed in the automated planning community to estimate the utility of belief states. We implemented our domain independent heuristics into the rollout mechanism of POMCP — a well known online POMDP planner that constructs a search tree to evaluate which action to take next. We provide an empirical evaluation showing how heuristics provide much leverage, especially in complex domains that require a long planning horizon, compared to the standard uniform rollout policy that is often used in POMCP.

For future research we intend to integrate our methods into other solvers, such as RTDP-BEL, or into point-based planners as a method to gather good belief points. We can also experiment with additional heuristics, other than the h_{add} heuristic used in this paper.

Author Contributions O.B. : ideas, implementation, experiments. G.S. : ideas, implementation, writing.

Funding Open access funding provided by Ben-Gurion University.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Agrawal, Rajeev: Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. Adv. Appl. Probab. **27**(4), 1054–1078 (1995)
2. Alagoz, O., Hsu, H., Schaefer, A.J., Roberts, M.S.: Markov decision processes: a tool for sequential decision making under uncertainty. Med. Decision Making **30**(4), 474–483 (2010)
3. Albore, A., Geffner, H.: Acting in partially observable environments when achievement of the goal cannot be guaranteed. In: Workshop on Planning and Plan Execution for Real-World Systems, at 19th Int. Conf. on Planning and Scheduling (ICAPS '09). (2009)
4. Albore, A., Palacios, H., Geffner, H.: A translation-based approach to contingent planning. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 1623–1628 (2009)
5. Albore, A., Palacios, H., Geffner, H.: A translation-based approach to contingent planning. In: IJCAI, vol. 9. pp. 1623–1628 (2009). https://doi.org/10.5555/1661445.1661706, https://dl.acm.org/doi/10.5555/1661445.1661706
6. Albore, A., Ramírez, M., Geffner, H.: Effective heuristics and belief tracking for planning with incomplete information. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011. (2011)
7. Audibert, J.-Y., Munos, R., Szepesvári, C.: Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. Theoret. Comput. Sci. **410**(19), 1876–1902 (2009)
8. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**, 235–256 (2002)
9. Bellman, R.: Dynamic programming. Courier Corporation (1957)
10. Bellman, R.: A markovian decision process. J. Math. Mech. 679–684 (1957)

11. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Strong planning under partial observability. Artif. Intell. **170**(4–5), 337–384 (2006)
12. Bonet, B., Geffner, H.: Planning as heuristic search. Artif. Intell. **129**(1–2), 5–33 (2001). https://doi.org/10.1016/S0004-3702(01)00108-4
13. Bonet, B., Geffner, H.: Solving pomdps: Rtdp-bel vs. point-based algorithms. In: Boutilier, C., (ed) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 1641–1646 (2009). https://doi.org/10.5555/1661445.1661709, https://dl.acm.org/doi/10.5555/1661445.1661709
14. Bonet, B., Geffner, H.: Planning under partial observability by classical replanning: theory and experiments. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 1936–1941 (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-324
15. Bonet, B., Geffner, H.: action selection for mdps: anytime ao* versus uct. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 26. pp. 1749–1755 (2012)
16. Brafman, R.I., Shani, G.: A multi-path compilation approach to contingent planning. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada. (2012)
17. Brafman, R.I., Shani, G.: Replanning in domains with partial information and sensing actions. J. Artif. Intell. Res. **45**, 565–600 (2012). https://doi.org/10.1613/jair.3711
18. Brafman, R.I., Shani, G.: Online belief tracking using regression for contingent planning. Artif. Intell. **241**, 131–152 (2016). https://doi.org/10.1016/j.artint.2016.08.005
19. Cai, P., Luo, Y., Hsu, D., Lee, W.S.: Hyp-despot: a hybrid parallel algorithm for online planning under uncertainty. Int. J. Robotics Res. **40**(2–3), 558–573 (2021)
20. Cassandra, A.R., Littman, M.L., Zhang, N.L.: Incremental pruning: a simple, fast, exact method for partially observable markov decision processes. (2013). arXiv:1302.1525
21. Michael Chung. Monte carlo planning in rts games. (2005)
22. Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In: International conference on computers and games. pp. 72–83. Springer (2006)
23. Geffner, H., Bonet, B.: Solving large pomdps using real time dynamic programming. In Working Notes Fall AAAI Symposium on POMDPs, vol. 218. (1998)
24. Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Springer Nature (2022)
25. Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al.: Domain-independent construction of pattern database heuristics for cost-optimal planning. AAAI **7**, 1007–1012 (2007)
26. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: What's the difference anyway? In: Brim, L., Edelkamp, S., Hansen, E.A., Sanders, P. (eds) Graph Search Engineering, 29.11. - 04.12.2009, volume 09491 of Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2009). https://doi.org/10.1609/icaps.v19i1.13370, http://drops.dagstuhl.de/opus/volltexte/2010/2432/
27. Helmert, M., Haslum, P., Hoffmann, J., Nissim, R.: Merge-and-shrink abstraction: a method for generating lower bounds in factored state spaces. J. ACM **61**(3), 1–63 (2014). https://doi.org/10.1145/2559951
28. Hoffmann, J., Nebel, B.: The FF planning system: fast plan generation through heuristic search. JAIR **14**, 253–302 (2001). https://doi.org/10.1613/jair.855
29. Hoffmann, J., Brafman, R.I.: Contingent planning via heuristic forward search witn implicit belief states. In: Biundo, S., Myers, K.L., Rajan, K. (eds) Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA. pp. 71–80. AAAI (2005). https://doi.org/10.5555/3037062.3037072, http://www.aaai.org/Library/ICAPS/2005/icaps05-008.php
30. Hörger, M., Kurniawati, H., Elfes, V.: Multilevel monte-carlo for solving pomdps online. In: Asfour, T., Yoshida, E., Park, J., Christensen, H., Khatib, O. (eds) Robotics Research - The 19th International Symposium ISRR 2019, Hanoi, Vietnam, October 6-10, 2019, vol. 20 of Springer Proceedings in Advanced Robotics. pp. 174–190. Springer (2019). https://doi.org/10.1007/978-3-030-95459-8_11
31. Howard, R.A.: Information value theory. IEEE Transactions on systems science and cybernetics **2**(1), 22–26 (1966). https://doi.org/10.1109/TSSC.1966.300074
32. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1–2), 99–134 (1998)
33. Keller, T., Eyerich, P.: PROST: probabilistic planning based on UCT. In: McCluskey, L., Williams, B.C., Silva, J.R., Bonet, B. (eds) Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012. AAAI (2012).

https://doi.org/10.1609/icaps.v22i1.13518, http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715
34. Keller, T., Helmert, M.: Trial-based heuristic tree search for finite horizon mdps. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 23, pages 135–143, 2013. https://doi.org/10.1609/icaps.v23i1.13557, http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6026
35. Kim, S.-K., Salzman, O., Likhachev, M.: Pomhdp: search-based belief space planning using multiple heuristics. In: Proceedings of the International Conference on Automated Planning and Scheduling, volume 29. pp. 734–744. (2019). https://doi.org/10.1609/icaps.v29i1.3542, https://ojs.aaai.org/index.php/ICAPS/article/view/3542
36. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning. pp. 282–293. Springer (2006)
37. Kurniawati, H.: Partially observable markov decision processes and robotics. Ann. Rev. Control Robot. Auto. Syst. **5**, 253–277 (2022). https://doi.org/10.1146/annurev-control-042920-092451
38. Kurniawati, H., Yadav, V.: An online pomdp solver for uncertainty planning in dynamic environment. In: Robotics Research: The 16th International Symposium ISRR. pp. 611–629. Springer (2016). https://doi.org/10.1007/978-3-319-28872-7_35
39. Kurniawati, H., Hsu, D., Lee, W.S.: Sarsop: efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems, vol. 2008. Citeseer (2008)
40. Luo, Y., Bai, H., Hsu, D., Lee, W.S.: Importance sampling for online planning under uncertainty. Int. J Robot. Res. **38**(2–3), 162–181 (2019). https://doi.org/10.1177/0278364918780322
41. Muise, C.J., Belle, V., McIlraith, S.A.: Computing contingent plans via fully observable non-deterministic planning. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 2322–2329 (2014)
42. Pineau, J., Gordon, G., Thrun, S., et al.: Point-based value iteration: an anytime algorithm for pomdps. Ijcai **3**, 1025–1032 (2003)
43. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
44. Richter, S., Helmert, M., Westphal, M.: Landmarks revisited. AAAI **8**, 975–982 (2008) http://www.aaai.org/Library/AAAI/2008/aaai08-155.php
45. J. Rintanen: Regression for classical and nondeterministic planning. In: ECAI 2008. pp. 568–572. IOS Press (2008). https://doi.org/10.3233/978-1-58603-891-5-568
46. Ross, S., Pineau, J., Paquet, S., Chaib-Draa, B.: Online planning algorithms for pomdps. J Artif. Intell. Res. **32**, 663–704 (2008). https://doi.org/10.1613/jair.2567
47. Saborío, J.C., Hertzberg, J.: Planning under uncertainty through goal-driven action selection. In: Agents and Artificial Intelligence: 10th International Conference, ICAART 2018, Funchal, Madeira, Portugal, January 16–18, 2018, Revised Selected Papers 10. pp. 182–201. Springer (2019). https://doi.org/10.1007/978-3-030-05453-3_9
48. Saborío, J.C., Hertzberg, J.: Efficient planning under uncertainty with incremental refinement. In: Uncertainty in Artificial Intelligence. pp. 303–312. PMLR (2020). https://doi.org/10.1109/TSMC.1987.4309045
49. Shani, G., Brafman, R.I.: Replanning in domains with partial information and sensing actions. In: IJCAI, volume 2011. pp. 2021–2026. Citeseer (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-337
50. Shani, G., Brafman, R.I., Shimony, S.E.: Forward search value iteration for pomdps. In: IJCAI. pp. 2619–2624. Citeseer (2007)
51. Shani, G., Pineau, J., Kaplow, R.: A survey of point-based pomdp solvers. Auton. Agent. Multi-Agent Syst. **27**, 1–51 (2013). https://doi.org/10.1007/s10458-012-9200-2
52. Shen, W., Trevizan, F., Toyer, S., Thiébaux, S., Xie, L.: Guiding search with generalized policies for probabilistic planning. Proc. Int. Symp. Comb. Search **10**, 97–105 (2019). https://doi.org/10.1609/socs.v10i1.18507
53. Shen, W., Trevizan, F., Thiébaux, S.: Learning domain-independent planning heuristics with hypergraph networks. Proc Int. Conf. Autom. Plann. Sched. **30**, 574–584 (2020)
54. Silver, D., Veness, J.: Monte-carlo planning in large pomdps. Adv. Neural Inf. Process. Syst. 23 (2010). https://doi.org/10.5555/2997046.2997137
55. Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable markov processes over a finite horizon. Oper. Res. **21**(5), 1071–1088 (1973). https://doi.org/10.1287/opre.21.5.1071
56. Smith, T., Simmons, R.: Heuristic search value iteration for pomdps. (2012). arXiv:1207.4166
57. Somani, A., Ye, N., Hsu, D., Lee, W.S.: Despot: online pomdp planning with regularization. Adv. Neural Inf. Process Syst. 26 (2013). https://doi.org/10.1613/jair.5328

58. Sondik, E.J.: The optimal control of partially observable markov processes over the infinite horizon: discounted costs. Oper. Res. **26**(2), 282–304 (1978)
59. Spaan, M.T.J., Vlassis, N.: Perseus: randomized point-based value iteration for pomdps. J. Artif. Intell. Res. **24**, 195–220 (2005)
60. Sunberg, Z., Kochenderfer, M.: Online algorithms for pomdps with continuous state, action, and observation spaces. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 28. pp. 259–263 (2018). https://doi.org/10.48550/arXiv.1709.06196
61. Trunda, O., Barták, R.: Deep learning of heuristics for domain-independent planning. ICAART (2). pp. 79–88 (2020)
62. Xiao, Y., Katt, S., ten Pas, A., Chen, S., Amato, C.: Online planning for target object search in clutter under partial observability. 2019 International Conference on Robotics and Automation (ICRA). pp. 8241–8247. IEEE (2019). https://doi.org/10.1109/ICRA.2019.8793494
63. Ye, N., Somani, A., Hsu, D., Lee, W.S.: Despot: Online pomdp planning with regularization. J. Artif. Intell. Res. **58**, 231–266 (2017). https://doi.org/10.1613/jair.5328