Applied Intelligence
https://doi.org/10.1007/s10489-018-1248-5

Posterior sampling for Monte Carlo planning under uncertainty

Aijun Bai[1] · **Feng Wu**[2] · **Xiaoping Chen**[2]

Abstract
Monte Carlo tree search (MCTS) has recently been drawing great interest in the domain of planning and learning under uncertainty. One of the fundamental challenges is the trade-off between exploration and exploitation. To address this problem, we propose to balance between exploration and exploitation via posterior sampling in the contexts of Markov decision process (MDP) and partially observable Markov decision process (POMDP). Specifically, we treat the cumulative reward returned by taking an action from a search node in the MCTS search tree as a random variable following an unknown distribution. We parametrize this distribution by introducing necessary hidden parameters, and infer the posterior distribution of the hidden parameters in a Bayesian way. We further expand a node in the search tree by using Thompson sampling to select an action based on its posterior probability of being optimal. Following this idea, we develop *Dirichlet-NormalGamma based Monte Carlo tree search* (DNG-MCTS) and *Dirichlet-Dirichlet-NormalGamma based partially observable Monte Carlo planning* (D^{2}NG-POMCP) algorithms respectively for Monte Carlo planning in MDPs and POMDPs. Experimental results show that the proposed algorithms outperform the state-of-the-art with better values on several benchmark problems.

Keywords Monte Carlo tree search · Thompson sampling · Planning under uncertainty

1 Introduction

The general task of sequential decision-making under uncertainty is of great interest to the *Artificial Intelligence* (AI) community. It embraces a broad range of common problems found in planning and learning. Currently, the most general and clear fundamental formulations for these problems are achieved through the theories of *Markov decision processes* (MDPs) and *Partially observable Markov decision processes* (POMDPs). MDP provides a rich mathematical framework for planning and learning under uncertainty in fully observable environments [64]; POMDP extends MDP to partially observable environments [46]. For many real-world problems (e.g. robotics and computer games to name a few), fully specified MDP/POMDP models represented as transition and reward functions are not easy to obtain. However, simulators (a.k.a. *generative* models) are usually available, or are relatively easy to develop. In this paper, we consider the problems of approximately solving MDPs and POMDPs online given only the underlying simulators via *Monte Carlo tree search* (MCTS) [20]. In the context of planning and learning under uncertainty, the key idea of MCTS is to evaluate each tree node (i.e., a state for an MDP or a belief state for a POMDP) using sampled trajectories starting from that node. It eventuelly finds a near-optimal policy by building a best-first search tree based on Monte Carlo simulation. MCTS is a model-free method and requires only a generative model of the underlying problem. MCTS has shown to be computationally efficient, anytime and highly parallelisable. To date, great success has been achieved by MCTS in variety of domains, such as playing games [37, 68, 69, 81], planning under uncertainty [8, 52, 70], and Bayesian reinforcement learning [5, 42, 78].

One of the fundamental challenges of MCTS is the well-known *exploration vs. exploitation* dilemma. That is to say, when building the search tree, the agent must not only exploit by selecting the action that currently seems

✉ Feng Wu
wufeng02@ustc.edu.cn

Aijun Bai
aijun.bai@microsoft.com

Xiaoping Chen
xpchen@ustc.edu.cn

[1] Cloud & AI One Microsoft Way, Redmond, WA 98052, USA

[2] University of Science and Technology of China, 96 Jinzhai Road, Hefei, Anhui 230026, China

Published online: 15 August 2018

best, but should also keep exploring for possible higher future pay-offs [47, 73]. Expanding the action that currently seems best can keep the search focused in known promising area, but could also miss better actions that have not been explored sufficiently. On the other hand, too much exploring on sub-optimal actions wastes significant portion of limited computational resources. UCB1 is probably the most successful and widely-used algorithm to address this dilemma [6, 7]. It is originally introduced in the *multi-armed bandit* problems (MABs) [55]. An algorithm for an MAB must decide which action (namely an *arm*) to apply at each time step, based on the outcomes of the previous plays. UCB1 selects the action that maximizes the UCB1 heuristic which defines an *upper confidence bound* (UCB) for the underlying action-values. Auer et al. [7] prove that UCB1 is asymptotically optimal for MABs. On the other hand, Thompson sampling is perhaps one of the earliest heuristics that tackles this problem in a Bayesian fashion according to the principle of *randomized probability matching* [75]. It stochastically selects an action based on its posterior probability of being optimal. Comparing to UCB1, one of the main advantages of Thompson sampling is that it can handle a wide range of information models with prior and posterior distributions, which go beyond using the expectations of reward alone [38].

In MABs, there usually exist two assessment criterions: a *cumulative regret* defined as the difference between the sum of the best action's expected reward and the obtained reward of actual actions applied so far, and a *simple regret* defined as the difference between the best expected reward and the expected reward of the action with highest sample mean among all actions currently. The cumulative regret is suitable for the cases when an algorithm tries to optimize the long-term total reward by trading off between exploration and exploitation appropriately [21]. The simple regret makes more sense for algorithms focusing on pure explorations of action pulls where only the last action collects a reward [22]. Apart from the fact that Thompson sampling empirically converges faster in terms of cumulative regret than the UCB1 approach [24], it has recently been proved that Thompson sampling achieves logarithmic cumulative regret which is asymptotically optimal for MABs [1, 2, 48, 53]. In Monte Carlo online planning, it is usually the final action actually applied to the environment that collects a reward. Therefore, it is more reasonable to minimize the simple regret instead of the cumulative regret in the context of Monte Carlo online planning [31]. However, simple and cumulative regrets can not be minimized simultaneously; moreover, Bubeck et al. [21] shows that in many cases the smaller the cumulative regret, the greater the simple regret. A recently growing understanding is that it is better to balance between cumulative and simple regrets in MCTS [77], since although the algorithm does not collect a real reward when searching the tree, it is good to grow the asymmetric tree more accurately by moderately exploiting the current tree. The reduction rate in terms of simple regret for Thompson sampling remains an open question, however, it is our observation that Thompson sampling empirically appears to have lower simple regret than the state-of-the-art, particularly for larger action space. This motivates us to apply Thompson sampling on MCTS for MDPs and POMDPs as it seems to be a promising approach in handling both cumulative and simple regrets.

Our work is motivated by many real-world problems that can be modeled as POMDPs. One of such examples is the well-known Canadian traveller problem (CTP)[1]. For the applicability of our settings, consider a robot navigating outdoors (e.g., a search and rescue robot in disaster response) equipped with an overhead map of the surrounding area (generated by satellite or an aerial vehicle). The resolution of the map may be much lower than the resolution used by the robot to navigate. Due to this low resolution, there is some uncertainty as to whether portions of the terrain are actually traversable or not, representing by some prior probabilities. Those incomplete information may be crucial for the robot's tasks. In such problems, the robot usually uses a generative model (a.k.a., a simulator[2]), which simulates the uncertainty of the environment, when computing the best path. Note that the up-to-date information is also incorporated in the model as the robot moves on. Other applications include vessel routing in the presence of uncertain weather conditions and the routing of automobiles in the presence of partially known and stochastically changing road congestion levels, where automated driver assistance must reason about the uncertainty with some prior knowledge and find the best path in the environment involving random events and incomplete information.

In this paper, we borrow the idea of Thompson sampling and propose novel Bayesian posterior sampling approaches to Monte Carlo based online planning in MDPs and POMDPs under the assumption that only simulators are available in advance. Specifically, based on our previous effort [10, 12], we develop *Dirichlet-NormalGamma based Monte Carlo tree search* (DNG-MCTS) and *Dirichlet-Dirichlet-NormalGamma based partially observable Monte Carlo planning* (D^2NG-POMCP) algorithms for MDPs and POMDPs respectively. In DNG-MCTS, we use a mixture of Normal distributions to model the unknown distribution of the cumulative reward of performing a particular action

[1] More details about the CTP problem is in Section 2.1.

[2] Discussion on the advantage of using a simulator is in Section 6.1.

in the MCTS search tree. We show that, in presence of online planning for MDPs, a conjugate prior exists in the form of a combination of Dirichlet and NormalGamma distributions. By choosing the conjugate prior, it is then relatively simple to compute the posterior distribution after each cumulative reward has been observed by simulation in the search process. Thompson sampling is then used to select the action to be performed by simulation at each decision node. The basic assumptions of DNG-MCTS are made due to the fact that, given a policy, an MDP reduces to a Markov chain defined over the state space. Unfortunately, this cannot be straightforwardly extended to POMDPs, because the Markov chain of a POMDP with given policy must be defined over the joint space of the state and belief space. Therefore, different assumptions are critically required to use posterior sampling techniques in POMDPs. Accordingly, in D^{2}NG-POMCP, we represent the uncertainty of the immediate reward as a Multinomial distribution (which is known for MDPs in contrast) and the cumulative reward returned by performing an action from a belief state in the search tree as a convex combination of Normal mixtures. We perform statistical inference on the posterior distribution in Bayesian settings by choosing the conjugate prior in the form of a combination of two Dirichlet and one NormalGamma distributions. We then use Thompson sampling to select an action to be performed by simulation at each decision node. We have tested the resulting algorithms on several benchmark problems. Experimental results confirm that our algorithms outperform the state-of-the-art online planning methods. Furthermore, we show the convergence properties of the proposed algorithms confirming the technical soundness.

The reminder of this paper is organized as follows. In Section 2, we briefly review the background. Section 3 introduces some related work. Sections 4 and 5 present the proposed algorithms in detail. Section 6 discusses on how to choose prior distributions, and shows the convergence property of the algorithms. We show experimental results on several benchmark problems in Section 7. And in Section 8, we conclude with a summary of our contributions and future work.

2 Background

We briefly review the CTP problem, the MDP and POMDP models, the MAB problem and the MCTS framework, as well as the UCT and POMCP algorithms.

2.1 CTP

In the CTP problem, a Canadian driver must travel a network of cities to her destination facing the situation that snowfall randomly blocks roads[3]. For the driver, the uncertainty is due to the fact that some roads may be blocked by snowfall so she must explore other paths towards her goal. In this scenario, the event of snowfall blocking roads is stochastic and can be observed only when she reaches those roads. The objective is then for the driver to reach her destination as fast as possible with minimum energy and time consumptions. In this case, the cumulative reward of selecting a road follows an unknown distribution depending on whether this road and the follow-up roads are blocked. For such problem, it is generally straightforward to design a simulator to simulate the travel on a city network with snowfall randomly blocking roads. Yet in the planning phase (i.e. the driver's thinking phase when playing with the simulator), being blocked by a certain road doesn't affect the final performance when acting. In fact, being blocked during simulation would instead reveal some hidden structure or information about the underlying city network and its optimal travel policy, which should be exploited in following simulations and the final action phase. The real objective in simulation is not the final cumulative reward that the driver can collect during the action phase, but the so-called simple reward. But optimizing simple reward directly almost equals *pure exploration* without taking advantage of results that have already been known during simulation, and is harmful during the long run. In this paper, we apply Thompson sampling in MCTS which appears to balance well between pure exploration and exploitation. More details about the problem settings can be found in Section 7.

2.2 MDP

Formally, an MDP is defined as a 4-tuple $\langle S, A, T, R\rangle$, where S is the state space, A is the action space, $T(s' \mid s, a)$ is the probability of reaching state s' after having applied action a in state s, and $R(s, a)$ is the immediate reward received. A *policy* $\pi : S \rightarrow A$ of an MDP is a mapping from states to actions, with $\pi(s)$ specifying the action which should be taken in state s. Given a policy π, the expected total reward of a state by following π (also known as the value function) is defined as:

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(s_t)\right)\right], \tag{1}$$

where $\gamma \in (0, 1]$ is a discount factor, s_t is the state in time step t and $\pi(s_t)$ is the action selected by policy π in state s_t. The aim of solving an MDP is to find the *optimal* policy π^* that maximizes the value function for all states. The

[3] https://en.wikipedia.org/wiki/Canadian_traveller_problem

respective optimal value function, denoted by V^*, satisfies the famous Bellman equation [16]:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^*(s') \right\}. \quad (2)$$

2.3 POMDP

A POMDP is an extension of MDP to partially observable environments, which is defined as a 6-tuple $\langle S, A, O, T, \Omega, R \rangle$, where S, A, T and R remain the same meanings as in MDPs, O is the observation space, and $\Omega(o \mid s,a)$ is the probability of observing o after having performed action a and reached state s. A POMDP can be transformed into an MDP over *belief state* (or *belief* for short) space. A belief b is a sufficient statistic for the history of actions and observations, defined as a probability distribution over the state space, with $b(s)$ denoting the probability of being in state s. Given an initial belief b_0, a history of action-observation pairs $h = (a_0, o_1, a_1, o_2, \ldots a_{t-1}, o_t)$ uniquely determines the resulting belief, which can be obtained by recursively using a Bayesian filter $b' = \zeta(b,a,o)$, written as:

$$b'(s') = \eta \Omega(o \mid s', a) \sum_{s \in S} T(s' \mid s,a) b(s), \quad (3)$$

where $\eta = 1/P(o \mid b,a)$ is a normalizing constant.

Let $\mathcal{B}$ be the set of all possible beliefs, a policy $\pi : \mathcal{B} \to A$ of a POMDP is defined as a mapping from belief space to actions. The goal of solving a POMDP is to find the optimal policy that maximizes the expected total reward for any beliefs. A POMDP can be transformed to a *Bayesian-adaptive MDP* (BAMDP, also known as a *belief MDP*): $\langle \mathcal{B}, A, T^+, r \rangle$, where $\mathcal{B}$ is the state space, A is the action space, $r(b,a) = \sum_{s \in S} b(s) R(s,a)$ is the reward function, and T^+ is the transition function, defined as:

$$T^+(b' \mid b,a) = \sum_{o \in O} \mathbf{1}[b' = \zeta(b,a,o)] \Omega(o \mid b,a), \quad (4)$$

where $\mathbf{1}$ is the indicator function. The Bellman equation of the resulting MDP is:

$$V^*(b) = \max_{a \in A} \left\{ r(b,a) + \gamma \sum_{o \in O} \Omega(o \mid b,a) V^*(\zeta(b,a,o)) \right\}. \quad (5)$$

Given an initial belief, the terms *belief* and *history* can be used interchangeably. In this paper, we formally present our main results in terms of belief, but implement the algorithm with respect to histories instead.

2.4 MAB

MABs are usually seen as fundamental decision-making components of planning and learning problems. Intuitively, an MAB can be seen as an MDP with only one state s and a stochastic reward function $R(s,a) := X_a$, where X_a is a random variable following an unknown stationary distribution $f_{X_a}(x)$. At each time step t, one action a_t must be chosen and executed. A stochastic reward X_{a_t} is then observed. The goal of solving an MAB is usually to find a policy that minimizes the cumulative regret defined as

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} (X_{a^*} - X_{a_t}) \right], \quad (6)$$

where a^* is the oracle best action. A simple regret defined for a pure exploration strategy after n times of action pulls is:

$$r_n = \mathbb{E}\left[X_{a^*} - X_{\bar{a}} \right], \quad (7)$$

where $\bar{a} = \text{argmax}_{a \in A} \bar{X}_a$ is the action with maximal empirical mean of reward.

2.5 MCTS

In the domain of online planning for MDPs and POMDPs, MCTS generally evaluates a node (i.e., a state in MDPs or a belief state in POMDPs) in the search tree by: 1) selecting an action according to an *action-selection* strategy; 2) performing the selected action by Monte Carlo simulation; 3) recursively evaluating the resulting state/belief if it is already in the search tree, or inserting it into the search tree and playing a *rollout policy* by Monte Carlo simulations; and 4) updating the statistics of tree nodes by back-propagating the simulation results up to the root node [20, 25]. Iteratively repeating this process, MCTS builds an asymmetric best-first search tree simultaneously. When interrupted at any time, MCTS reports the best action based on current values of nodes in the search tree.

UCT In the context of Monte Carlo online planning for MDPs, *UCB applied to trees* (UCT) perhaps is one of the most popular implementations of MCTS [3, 4, 14, 32, 34, 36, 37, 50, 52, 81, 82]. It treats each state of the search tree as an MAB, and selects the action that maximizes the UCB1 heuristic, defined as:

$$\text{UCB1}(s,a) = \bar{Q}(s,a) + c \sqrt{\frac{\log N(s)}{N(s,a)}}, \quad (8)$$

where $\bar{Q}(s,a)$ is the mean return of action a in state s from all previous simulations, $N(s,a)$ is the visitation count of action a in state s, $N(s) = \sum_{a \in A} N(s,a)$ is the overall count, and c is a constant that determines the relative ratio of exploration to exploitation. Kocsis et al. [52] proved that

with an appropriate choice of c the probability of selecting the optimal action converges to 1 as the number of samples grows to infinity.

POMCP *Partially observable Monte Carlo planning* (POMCP) is an extension of UCT to POMDPs [70]. POMCP employs a *root sampling* technique to start the search from a state sampled from belief $b(h)$ associated with history h of the root node, making the usage of only a state-based simulator possible. At each decision node, POMCP chooses the action that maximizes the UCB1 heuristic, defined in terms of histories and actions:

$$\text{UCB1}(h,a) = \bar{Q}(h,a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}, \tag{9}$$

where $\bar{Q}(h,a)$ is the average outcome of applying action a in history node h over all previous simulations, $N(h,a)$ is the visitation count of action a following h, $N(h) = \sum_{a\in A} N(h,a)$ is the overall count, and c is the exploration constant. POMCP uses a particle filter [39, 76] to approximate the belief state, by adapting a Monte Carlo procedure to update particles based on sampled observations, rewards, and state transitions. Silver et al. [70] show that, for a suitable choice of c, the value function constructed by POMCP with the root sampling technique converges in probability to the optimal value function. POMCP has been shown with success in various problems [13, 56, 78].

3 Related work

In the context of MDP online planning, *real-time dynamic programming* (RTDP) [15, 18, 58, 67] is among the first that tries to find the "best" action for the current state by conducting a trial-based search process with greedy action selection and an admissible heuristic. Instead of trial-based search, AO* [19, 33, 43] builds an optimal solution graph with respect to the AND-OR graph by greedily expanding tip nodes in the current best partial solution graph and assigning values to new nodes according to an admissible heuristic function. MAXQ-OP finds the best action efficiently by exploiting the underlying hierarchical structure via MAXQ decomposition of the original MDP [9, 11]. MCTS finds near-optimal policies by combining tree search methods with Monte Carlo sampling techniques [10, 20, 31, 37, 49, 52]. Most recently, *trial-based heuristic tree search* (THTS) [51] was proposed to subsume these approaches by classifying five ingredients including heuristic function, backup function, action selection, outcome selection, and trial length.

The fundamental assumption of DNG-MCTS is to model the unknown distribution of the cumulative reward of applying an action in a state as a mixture of Normal distributions. A similar assumption has been made in [27], where they assumed a Normal distribution over the rewards. Comparing with their approach, as we will show in Section 4, our assumption on Normal mixture is more realistic according to the central limit theorem on Markov chains. Tesauro et al. [74] develops a Bayesian UCT approach to MCTS using Gaussian approximation. Specifically, their method propagates probability distributions of rewards from leaf nodes up to the root node by applying MAX (or MIN) extremum distribution operator for the interior nodes. Then, it uses modified UCB1 heuristics to select actions on the basis of the interior distributions. However, extremum distribution operation is very time consuming since it must consider over all the child nodes. In contrast, we treat each decision node in the search tree as an MAB, maintain a posterior distribution over the cumulative reward for each applicable actions separately, and then select the best action using Thompson sampling.

Online planning methods for POMDPs aim to alleviate the complexity of computing a full policy by planning only for the current belief state [66]. Due to the differences of expanding the current belief state, online planning methods can be roughly classified into three categories: *branch-and-bound pruning*, *heuristic search* and *Monte Carlo sampling*. Branch-and-bound pruning technique prunes nodes that are known to be suboptimal, thus preventing the expansion of unnecessary branches of the search tree [40, 41, 61, 62]. Heuristic search algorithms try to focus the search on the most relevant reachable beliefs by using heuristics to select the most promising belief nodes to expand [54, 65, 71, 72, 80, 83]. Monte Carlo sampling technique reduces the branching factor by sampling one or more observations to conduct the search process, allowing for deeper search within a set planning time [17, 23, 57, 70, 76].

In the context of reinforcement learning, Wang et al. [79] developed a posterior sampling technique for approximating optimal decision-making for Bayesian reinforcement learning. Osband et al. [59] extend the similar idea to more general reinforcement learning problems. Specifically, their approaches maintain a posterior distribution over MDP models (i.e., the transition and reward functions), sample an MDP, and solve the sampled MDP to select an action for current decision-node. However, their methods require to repeatedly solve the sampled MDP for each action selected, which is very time consuming for large problems. In contrast, our method directly maintains posterior distributions of action values and selects an action based on its posterior probability of being optimal.

4 Posterior sampling based Monte Carlo planning for MDPs

This section presents the proposed Bayesian posterior sampling algorithm for Monte Carlo online planning in MDPs, namely DNG-MCTS.

4.1 Assumptions

We base our assumptions on the central limit theorem on Markov chains [26, 45].

Theorem 1 (Central limit theorem on Markov chains) *Let $X = \{x_0, x_1, \dots\}$ be an ergodic Markov chain on a countable state space $\mathcal{X}$ with stationary distribution w having support $\mathcal{X}$. Let f be any bounded function defined over $\mathcal{X}$, and define $\mu = \mathbb{E}_w[f] = \int_{\mathcal{X}} w(x) f(x) \mathrm{d}x$, and $\sigma = Var_w(f(x_0)) + 2\sum_{i=1}^{\infty} Cov_w(f(x_0), f(x_i))$. Let $\mathcal{N}(0, \sigma^2)$ be a Normal distribution, then for any initial distribution of x_0, as $n \to \infty$, we have:*

$$\sqrt{n}\left(\frac{1}{n}\sum_{t=0}^{n} f(x_t) - \mu\right) \to \mathcal{N}(0, \sigma^2). \tag{10}$$

Corollary 1 *Theorem 1 indicates that the sample mean $\frac{1}{n}\sum_{t=0}^{n} f(s_t)$ follows $\mathcal{N}(\mu, \sigma^2/n)$ as n grows to infinity. It is then natural to approximate the distribution of $\frac{1}{n}\sum_{t=0}^{n} f(s_t)$ as a Normal distribution if n is sufficiently large. Under this approximation, the sum $\sum_{t=0}^{n} f(s_t)$ is also following a Normal distribution, since n is a constant.*

For a given MDP policy π, let $X_{s,\pi}$ be a random variable that denotes the cumulative reward of following policy π starting from state s, and let $X_{s,a,\pi}$ denotes the cumulative reward of first performing action a in state s and then following policy π thereafter. Our assumptions are:

Assumption 1 $X_{s,\pi}$ is following a Normal distribution.

Assumption 2 $X_{s,a,\pi}$ is following a mixture of Normal distributions.

These are realistic approximations for our problems with the following reasons. Given policy π, an MDP reduces to a Markov chain $\{s_t\}$ defined over a finite state space S with transition function $P(s' \mid s) = T\left(s' \mid s, \pi(s)\right)$. Suppose the resulting Markov chain $\{s_t\}$ is ergodic, i.e., it is possible to go from every state to every other state (not necessarily in one move). For finite-horizon MDPs with planning horizon H, if $\gamma = 1$, $X_{s_0,\pi} = \sum_{t=0}^{H} R(s_t, \pi(s_t))$ is a sum of $f(s_t) = R(s_t, \pi(s_t))$. According to Corollary 1, $X_{s_0,\pi}$ is approximately normally distributed for each $s_0 \in S$ if H is sufficiently large. In the case, if $\gamma \neq 1$, it is still fairly reasonable to approximate $X_{s_0,\pi}$ as a Normal distribution, if H is sufficiently large and γ is close to 1.

If the policy π is not fixed and may change over time (e.g., the derived policy of an online algorithm before it converges), the real distribution of $X_{s,\pi}$ is actually unknown and could be very complex. However, if the algorithm is guaranteed to converge in the limit (as discussed in Section 6.3, this holds for the proposed DNG-MCTS algorithm), it is convenient and reasonable to approximate $X_{s,\pi}$ as a Normal distribution.

Now consider the cumulative reward of performing action a in state s and following policy π thereafter. By definition,

$$X_{s,a,\pi} = R(s, a) + \gamma X_{s',\pi}, \tag{11}$$

where s' is the next state distributed according to $T(s' \mid s, a)$. Let $Y_{s,a,\pi}$ be a random variable defined as:

$$Y_{s,a,\pi} = \frac{1}{\gamma}\left(X_{s,a,\pi} - R(s, a)\right). \tag{12}$$

It follows that the *probability density function* (p.d.f.) of $Y_{s,a,\pi}$ is a convex combination of the p.d.f.'s of $X_{s',\pi}$ for all $s' \in S$. Formally, we have:

$$f_{Y_{s,a,\pi}}(x) = \sum_{s' \in S} T(s' \mid s, a) f_{X_{s',\pi}}(x). \tag{13}$$

Hence it is straightforward to model the distribution of $Y_{s,a,\pi}$ as a mixture of Normal distributions, if $X_{s',\pi}$ is assumed to be normally distributed for each $s' \in S$. Since $X_{s,a,\pi}$ is a linear function of $Y_{s,a,\pi}$, $X_{s,a,\pi}$ is also following a mixture of Normal distributions under our assumptions.

4.2 Bayesian modeling and inference

Theorem 2 (Bayesian inference) *Suppose the unknown distribution of a random variable X is modeled as a parametric likelihood function $L(x \mid \theta)$ depending on some parameters θ. Let the prior distribution of θ be $P(\theta)$, After observing a set of independent and identically distributed samples $Z = \{x_1, x_2, \dots\}$ from the distribution of X, the posterior distribution of θ can then be obtained by applying Bayes' rule:*

$$P(\theta \mid Z) = \eta P(Z \mid \theta) P(\theta) = \eta \prod_i L(x_i \mid \theta) P(\theta), \tag{14}$$

where $\eta = 1/P(Z)$ is a normalizing constant.

Theorem 2 and Assumption 1 indicate that it suffices to model the distribution of $X_{s,\pi}$ using a Normal likelihood $\mathcal{N}(\mu_s, 1/\tau_s)$ with unknown mean μ_s and precision τ_s in Bayesian settings. The precision is defined as the reciprocal of the variance, such that $\tau = 1/\sigma^2$. This is chosen for mathematical convenience of introducing a NormalGamma distribution as a conjugate prior [28].

Definition 1 (NormalGamma distribution) A NormalGamma distribution is fully determined by a tuple of hyper-parameters $\langle \mu_0, \lambda, \alpha, \beta \rangle$, where $\lambda > 0$, $\alpha \geq 1$ and $\beta \geq 0$. Let $\Gamma(\cdot)$ be the gamma function, it is said that (μ, τ) follows a NormalGamma distribution $NormalGamma(\mu_0, \lambda, \alpha, \beta)$, if the p.d.f. of (μ, τ) has the form:

$$f(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) = \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}}\, \tau^{\alpha-\frac{1}{2}}\, e^{-\beta\tau}\, e^{-\frac{\lambda\tau(\mu-\mu_0)^2}{2}}. \tag{15}$$

Let us briefly recall the posterior of (μ, τ). By definition, the marginal distribution over τ is a Gamma distribution, denoted by $\tau \sim Gamma(\alpha, \beta)$, and the conditional distribution over μ given τ is a Normal distribution, denoted by $\mu \sim \mathcal{N}(\mu_0, 1/(\lambda\tau))$.

Theorem 3 (Posterior distribution of NormalGamma parameters) *Suppose X is normally distributed with unknown mean μ and precision τ: $X \sim \mathcal{N}(\mu, 1/\tau)$, and the prior distribution of (μ, τ) has a NormalGamma distribution: $(\mu, \tau) \sim NormalGamma(\mu_0, \lambda_0, \alpha_0, \beta_0)$. After observing n independent and identically distributed samples of X, denoted by $\{x_1, x_2, \ldots, x_n\}$, let $\bar{x} = \sum_{i=1}^{n} x_i/n$ and $s = \sum_{i=1}^{n} (x_i - \bar{x})^2/n$ be the sample mean and variance respectively, according to Bayes' theorem, the posterior distribution of (μ, τ) is also a NormalGamma distribution, namely $(\mu, \tau) \sim NormalGamma(\mu_n, \lambda_n, \alpha_n, \beta_n)$, where:*

$$\mu_n = \frac{\lambda_0\mu_0 + n\bar{x}}{\lambda_0 + n}, \tag{16}$$

$$\lambda_n = \lambda_0 + n, \tag{17}$$

$$\alpha_n = \alpha_0 + \frac{n}{2}, \tag{18}$$

$$\beta_n = \beta_0 + \frac{1}{2}\left(ns + \frac{\lambda_0 n(\bar{x} - \mu_0)^2}{\lambda_0 + n}\right). \tag{19}$$

Based on Assumption 2, the distribution of $Y_{s,a,\pi}$ can be modeled as a Normal mixtures:

$$Y_{s,a,\pi} \sim \sum_{s' \in S} w_{s,a,s'} \mathcal{N}(\mu_{s'}, \frac{1}{\tau_{s'}}), \tag{20}$$

where $w_{s,a,s'} = T(s' \mid s, a)$ are the mixture weights such that $w_{s,a,s'} \geq 0$ and $\sum_{s' \in S} w_{s,a,s'} = 1$. Notice that $w_{s,a,s'}$ are previously unknown in Monte Carlo online planning settings. A natural representation on these unknown weights is via Dirichlet distributions, since Dirichlet distribution is the conjugate prior of a general discrete probability distribution [28]. For state s and action a, a Dirichlet distribution, denoted by $Dirichlet(\boldsymbol{\rho}_{s,a})$, where $\boldsymbol{\rho}_{s,a} = (\rho_{s,a,s_1}, \rho_{s,a,s_2}, \ldots)$ is a vector of hyper-parameters, gives the posterior distribution of $T(s' \mid s, a)$ for each $s' \in S$ if the transition from (s, a) to s' has been observed $\rho_{s,a,s'} - 1$ times. It follows that, after observing a new transition $(s, a) \rightarrow s'$, the posterior distribution is also a Dirichlet, which can simply be updated as:

$$\rho_{s,a,s'} \leftarrow \rho_{s,a,s'} + 1. \tag{21}$$

Therefore, to model the distribution of $X_{s,\pi}$ and $X_{s,a,\pi}$, we need only to maintain a set of hyper-parameters $\langle \mu_{s,0}, \lambda_s, \alpha_s, \beta_s \rangle$ and $\boldsymbol{\rho}_{s,a}$ for all states s and state-action pairs (s, a) encountered in the MCTS search tree and update them by using Bayes' rule.

4.3 Thompson sampling based action selection

Definition 2 (Thompson sampling) Thompson sampling stochastically selects an action based on its posterior probability of being optimal. Formally, in Bayesian settings, action a is chosen with probability:

$$P(a) = \int \mathbf{1}\left[a = \underset{a'}{\operatorname{argmax}}\, \mathbb{E}\left[X_{a'} \mid \theta_{a'}\right]\right] \prod_{a'} P_{a'}(\theta_{a'} \mid Z)\, \mathrm{d}\boldsymbol{\theta}, \tag{22}$$

where θ_a is the hidden parameter (or a set of hidden parameters) specifying the underlying reward distribution of applying a (i.e., the distribution of X_a), $\boldsymbol{\theta} = (\theta_{a_1}, \theta_{a_2}, \ldots)$ is a vector of parameters for all actions, and $\mathbb{E}[X_a \mid \theta_a] = \int x L_a(x \mid \theta_a)\mathrm{d}x$ is the expectation of X_a given θ_a.

Thompson sampling can efficiently be approached by sampling method. To this end, a set of parameters θ_a is sampled according to the posterior distributions $P_a(\theta_a \mid Z)$ for each $a \in A$, and the action with the highest expectation is finally selected, namely:

$$a^* = \underset{a}{\operatorname{argmax}}\, \mathbb{E}[X_a \mid \theta_a]. \tag{23}$$

In DNG-MCTS, we use Thompson sampling to guide the action selection at each decision node of the search tree. More precisely, at decision node s, let s' be the possible next state after executing an action a. We sample the mean $\mu_{s'}$ and the mixture weights $w_{s,a,s'}$ according to $NormalGamma(\mu_{s',0}, \lambda_{s'}, \alpha_{s'}, \beta_{s'})$ and $Dirichlet(\boldsymbol{\rho}_{s,a})$ respectively. The sampled action-value $\tilde{Q}(s, a)$ is approximated as:

$$\tilde{Q}(s, a) = R(s, a) + \gamma \sum_{s' \in S} w_{s,a,s'} \mu_{s'}. \tag{24}$$

The action with the highest $\tilde{Q}$ value is then selected to perform by simulation.

4.4 DNG-MCTS

The main process of DNG-MCTS is outlined in Fig. 1. It is worth noticing that the function `ThompsonSampling` has a boolean parameter *sampling*. If *sampling* is true,

Fig. 1 Dirichlet-NormalGamma based Monte Carlo tree search

```
1  OnlinePlanning(s : state, T : tree)
2  Initialize H ← maximal planning horizon
3  repeat
4  |  DNG-MCTS(s, T, 0)
5  until resource budgets reached
6  return ThompsonSampling(s, 0, False)

7  DNG-MCTS(s : state, T : tree, d : depth)
8  if d ≥ H or s is terminal then
9  |  return 0
10 else if node ⟨s, d⟩ is not in tree T then
11 |  Initialize (μ_{s,0}, λ_s, α_s, β_s), and ρ_{s,a} for a ∈ A
12 |  Add node ⟨s, d⟩ to T
13 |  Play rollout policy by simulation for H − d steps
14 |  Get the cumulative reward r
15 |  return r
16 else
17 |  a ← ThompsonSampling(s, d, True)
18 |  Execute a by simulation
19 |  Observe next state s′ and reward R(s, a)
20 |  r ← R(s, a) + γ DNG-MCTS(s′, T, d + 1)
21 |  α_s ← α_s + 0.5
22 |  β_s ← β_s + (λ_s(r − μ_{s,0})^2/(λ_s + 1))/2
23 |  μ_{s,0} ← (λ_s μ_{s,0} + r)/(λ_s + 1)
24 |  λ_s ← λ_s + 1
25 |  ρ_{s,a,s′} ← ρ_{s,a,s′} + 1
26 |  return r
```

```
1  ThompsonSampling(s : state, d : depth, sampling : boolean)
2  foreach a ∈ A do
3  |  q_a ← QValue(s, a, d, sampling)
4  return argmax_a q_a

5  QValue(s : state, a : action, d : depth, sampling : boolean)
6  r ← 0
7  foreach s′ ∈ S do
8  |  if sampling = True then
9  |  |  Sample w_{s′} ∼ Dirichlet(ρ_{s,a})
10 |  else
11 |  |  w_{s′} ← ρ_{s,a,s′} / Σ_{s″∈S} ρ_{s,a,s″}
12 |  r ← r + w_{s′} Value(s′, d + 1, sampling)
13 r ← R(s, a) + γr
14 return r

15 Value(s : state, d : depth, sampling : boolean)
16 if d ≥ H or s is terminal then
17 |  return 0
18 else
19 |  if sampling = True then
20 |  |  Sample (μ, τ) ∼ NormalGamma(μ_{s,0}, λ_s, α_s, β_s)
21 |  |  return μ
22 |  else
23 |  |  return μ_{s,0}
```

Thompson sampling method is used to select the best action as explained in Section 4.3, otherwise a greedy action with the highest mean action-value $\bar{Q}(s, a)$ is returned, where:

$$\bar{Q}(s,a) = R(s,a) + \gamma \sum_{s' \in S} \frac{\rho_{s,a,s'}}{\sum_{s'' \in S} \rho_{s,a,s''}} \mu_{s',0}. \tag{25}$$

At each iteration, the **DNG-MCTS** function applies Thompson sampling to recursively select actions to be executed by simulation from the root node to leaf nodes through the existing search tree T. It inserts each newly visited node into the tree, plays a default rollout policy from the new node, and propagates the simulated outcome to update the hyper-parameters for visited states and actions. The **OnlinePlanning** function is the overall procedure interacting with the real environment. It is called with current state s, and search tree T initially empty. It repeatedly calls the **DNG-MCTS** function until some resource budgets are reached (e.g., the computation is timeout or the maximal number of iterations is reached). A greedy action to be performed in the environment is returned to the agent finally. Notice that the rollout policy is only played once for each new node at each iteration, the set of past observations Z in the algorithm has size $n = 1$.

5 Posterior sampling based Monte Carlo planning for POMDPs

In this section, we present the Bayesian posterior sampling algorithm for Monte Carlo planning in POMDPs, namely D^{2}NG-POMCP.

5.1 Assumptions

Suppose a POMDP agent following policy π is interacting with the environment, we treat $\langle s, b \rangle$ as a joint state, where s is the true state of the whole environment (including the agent) and b is the internal belief state of the agent. Let $\mathcal{J} = S \times \mathcal{B}$ be the joint space of the state space S and the belief space $\mathcal{B}$, the stochastic process of the joint state reduces to a Markov chain $\{\langle s_t, b_t \rangle\}$

defined over the joint space $\mathcal{J}$, with the transition function being:

$$P\left(\langle s', b'\rangle \mid \langle s, b\rangle\right) = T\left(s' \mid s, \pi(b)\right) T^{+}\left(b' \mid b, \pi(b)\right). \tag{26}$$

Let $X_{b,a}$ be a random variable denoting the immediate reward of performing action a in belief state b, let $X_{s,b,\pi}$ be a random variable that denotes the cumulative reward of following policy π from joint state $\langle s, b\rangle$, and let $X_{b,\pi}$ be a random variable denoting the cumulative reward of following policy π from belief state b. Our assumptions are:

Assumption 3 $X_{b,a}$ is following a Multinomial distribution.

Assumption 4 $X_{s,b,\pi}$ is following a Normal distribution.

Assumption 5 $X_{b,\pi}$ is following a mixture of Normal distributions.

We assume a discrete and finite set of possible immediate rewards in a POMDP, suppose the set is $\mathcal{I} = \{r_1, r_2, \ldots, r_k\}$, where $r_i = R(s, a)$ represents the immediate reward of taking action a in state s which is hidden. It is then easy to see that $X_{b,a}$ follows a *Multinomial distribution* (also known as a *categorical distribution*), denoted by $Multinomial(p_1, p_2, \ldots, p_k)$, such that $\sum_{i=1}^{k} p_i = 1$, with $p_i = \sum_{s \in S} \mathbf{1}[R(s, a) = r_i]b(s)$ being the probability of $X_{b,a} = r_i$ [35].

In POMDPs, the sub-space reachable from the initial belief state b_0 is countable, since each combination of historical action-observation pairs determines uniquely a resulting belief state, and the history space is naturally countable in lexicographic order. Therefore, the reachable sub-space from $\langle s_0, b_0\rangle$ is also countable, since the state space is countable by definition. For finite horizon POMDPs with planning horizon H, if $\gamma = 1$, $X_{s_0,b_0,\pi} = \sum_{t=0}^{H} R(s_t, \pi(b_t))$ can be seen as a sum of $f(s_t, b_t) = R(s_t, \pi(b_t))$. Suppose the resulting chain $\{\langle s_t, b_t\rangle\}$ is ergodic, according to Corollary 1, we claim that $X_{s_0,b_0,\pi}$ is approximately normally distributed for each $\langle s_0, b_0\rangle \in \mathcal{J}$, if H is sufficiently large. On the other hand, if $\gamma \neq 1$, but is close to 1, it is also reasonable to approximate $X_{s_0,b_0,\pi}$ as a Normal distribution, if H is sufficiently large. The cumulative reward of following π starting from belief b is completely determined in the reduced Markov chain provided with a stochastically sampled initial state s, i.e., $X_{b,\pi} = X_{s,b,\pi}$. Hence the p.d.f. of $X_{b,\pi}$ can be expressed as a convex combination of p.d.f.'s of $X_{s,b,\pi}$ by definition:

$$f_{X_{b,\pi}}(x) = \sum_{s \in S} b(s) f_{X_{s,b,\pi}}(x). \tag{27}$$

It is then straightforward to model the distribution of $X_{b,\pi}$ as a mixture of Normal distributions, if $X_{s,b,\pi}$ is assumed to be normally distributed for all $\langle s, b\rangle \in \mathcal{J}$.

In the case if the policy π is not fixed and changes over time, e.g., the derived policy of an online algorithm before it converges, the real distribution of $X_{b,\pi}$ is actually unknown and could be some kind of very complex distribution. However, if the algorithm is guaranteed to converge at infinity, it is then convenient and reasonable to approximate $X_{b,\pi}$ as a mixture of Normal distributions.

5.2 Bayesian modeling and inference

Assumption 3 implies that it suffices to model the distribution of $X_{b,a}$ as a Multinomial likelihood with unknown weights: $X_{b,a} \sim Multinomial(p_1, p_2, \ldots, p_k)$. A natural representation on these unknown weights is via Dirichlet distributions, since Dirichlet distribution is the conjugate prior of a Multinomial distribution. For belief state b and action a, the prior distribution of p_i is modeled as a Dirichlet distribution, denoted by $Dirichlet(\boldsymbol{\psi}_{b,a})$, where $\boldsymbol{\psi}_{b,a} = (\psi_{b,a,r_1}, \psi_{b,a,r_2}, \ldots, \psi_{b,a,r_k})$ is a vector of hyper-parameters. After observing an immediate reward r, the posterior distribution is also a Dirichlet, which is updated as:

$$\psi_{b,a,r} \leftarrow \psi_{b,a,r} + 1. \tag{28}$$

According to Assumption 4, we model the distribution of $X_{s,b,\pi}$ using a Normal likelihood $\mathcal{N}(\mu_{s,b}, 1/\tau_{s,b})$ with unknown mean $\mu_{s,b}$ and precision $\tau_{s,b}$. By choosing a NormalGamma distribution as a conjugate prior, the posterior distribution of $(\mu_{s,b}, \tau_{s,b})$ follows also a NormalGamma distribution, such that $(\mu_{s,b}, \tau_{s,b}) \sim NormalGamma(\mu_{s,b,0}, \lambda_{s,b}, \alpha_{s,b}, \beta_{s,b})$, where $\mu_{s,b,0}$, $\lambda_{s,b}$, $\alpha_{s,b}$, and $\beta_{s,b}$ are the hyper-parameters.

As explained in Assumption 5, $X_{b,\pi}$ follows a mixture of Normal distributions, which can be easily modeled as a convex combination of $b(s)$ and $X_{s,b,\pi}$ for $s \in S$. Now consider the cumulative reward of first performing action a in belief b and then following policy π thereafter, denoted by $X_{b,a,\pi}$. According to the definition,

$$X_{b,a,\pi} = X_{b,a} + \gamma X_{b',\pi}, \tag{29}$$

where b' is the next belief distributed according to $T^{+}(b' \mid b, a)$. It is difficult to explicitly describe the distribution of $X_{b,a,\pi}$. However, it is rather easy to compute the expectation, expressed as:

$$\begin{aligned}\mathbb{E}[X_{b,a,\pi}] &= \mathbb{E}[X_{b,a}] + \gamma \sum_{b' \in \mathcal{B}} \mathbb{E}[X_{b',\pi}] T^{+}(b' \mid b, a) \\ &= \mathbb{E}[X_{b,a}] + \gamma \sum_{o \in O} \mathbf{1}[b' = \zeta(b, a, o)] \Omega(o \mid b, a) \\ &\quad \mathbb{E}[X_{b',\pi}].\end{aligned} \tag{30}$$

In fact, if the underlying transition and observation functions are known, $\mathbb{E}[X_{b,a,\pi}]$ is usually defined as the Q action-value:

$$Q^{\pi}(b,a) = r(b,a) + \gamma \sum_{o \in O} \Omega(o \mid b,a) V^{\pi}(\zeta(b,a,o)). \tag{31}$$

Recall that in our assumptions, $X_{b,a}$ follows a Multinomial distribution, and $X_{b',\pi}$ follows a mixture of Normal distributions. The question turns to be how to model the previously unknown observation model — $\Omega(\cdot \mid b,a)$. Fortunately, $\Omega(\cdot \mid b,a)$ can be easily inferred in Bayesian settings by introducing a Dirichlet distribution as the conjugate prior, denoted by $Dirichlet(\boldsymbol{\rho}_{b,a})$, where $\boldsymbol{\rho}_{b,a} = (\rho_{b,a,o_1}, \rho_{b,a,o_2}, \dots)$ are the hyper-parameters. After observing a transition $(b,a) \rightarrow o$, the posterior distribution of $\Omega(\cdot \mid b,a)$ is updating as:

$$\rho_{b,a,o} \leftarrow \rho_{b,a,o} + 1. \tag{32}$$

Therefore, to compute the expectation of the posterior distribution of $X_{b,a,\pi}$, we only need to maintain a set of hyper-parameters $\langle \mu_{s,b,0}, \lambda_{s,b}, \alpha_{s,b}, \beta_{s,b} \rangle$, $\boldsymbol{\psi}_{b,a}$ and $\boldsymbol{\rho}_{b,a}$ for each state s, belief state b and action a encountered in the MCTS search tree, and update them by using Bayes' rule.

5.3 Thompson sampling based action selection

In D^{2}NG-POMCP, we use Thompson sampling to guide the action section at each decision node. More precisely, at the decision node associated with belief state b, to compute the expectation of $X_{b,a,\pi}$, we sample $w_{b,a,o}$ for $o \in O$ according to $Dirichlet(\boldsymbol{\rho}_{b,a})$, $w_{b,a,r}$ for $r \in \mathcal{I}$ according to $Dirichlet(\boldsymbol{\psi}_{b,a})$, and $\mu_{s',b'}$ for $\langle s', b' \rangle \in \mathcal{J}$ according to $NormalGamma(\mu_{s',b',0}, \lambda_{s',b'}, \alpha_{s',b'}, \beta_{s',b'})$, where $b' = \zeta(b,a,o)$ is the next belief after having performed action a and obtained observation o in belief b. Finally, the action with the highest sampled action-value $\tilde{Q}(b,a)$ is selected, where:

$$\begin{aligned} \tilde{Q}(b,a) &= \sum_{r \in \mathcal{I}} w_{b,a,r} r + \gamma \sum_{o \in O} \mathbf{1}[b' \\ &\quad = \zeta(b,a,o)] w_{b,a,o} \sum_{s' \in S} \mu_{s',b'} b'(s'). \end{aligned} \tag{33}$$

5.4 D^{2}NG-POMCP

The main process of D^{2}NG-POMCP is outlined in Fig. 2. As aforementioned, our algorithm is implemented on basis of histories instead of explicit beliefs. Given an initial belief, a history h uniquely determines the resulting belief, thus we maintain a set of hyper-parameters $\langle \mu_{s,h,0}, \lambda_{s,h}, \alpha_{s,h}, \beta_{s,h} \rangle$, $\boldsymbol{\psi}_{h,a}$ and $\boldsymbol{\rho}_{h,a}$ for each state s, history h and action a encountered in the MCTS search tree. Hence each node of the search tree is represented by a history h. Specifically, we use particles, denoted by $\mathcal{P}(h)$, to represent the respective belief state of history h, which are updated using a particle filter. Expressed in histories, (33) turns out to be:

$$\tilde{Q}(h,a) = \sum_{r \in \mathcal{I}} w_{h,a,r} r + \gamma \sum_{o \in O} w_{h,a,o} \sum_{s' \in \mathcal{P}(hao)} \mu_{s',hao}, \tag{34}$$

where $w_{h,a,r}$, $w_{h,a,o}$ and $\mu_{s',hao}$ are sampled randomly according to $Dirichlet(\boldsymbol{\psi}_{h,a})$, $Dirichlet(\boldsymbol{\rho}_{h,a})$ and $NormalGamma(\mu_{s',hao,0}, \lambda_{s',hao}, \alpha_{s',hao}, \beta_{s',hao})$ respectively. When the search process ended at the root of tree, a greedy action with the highest mean action-value $\bar{Q}(h,a)$ is selected, where:

$$\begin{aligned} \bar{Q}(h,a) &= \sum_{r \in \mathcal{I}} \frac{\psi_{h,a,r}}{\sum_{r' \in \mathcal{I}} \psi_{h,a,r'}} r \\ &\quad + \gamma \sum_{o \in O} \frac{\rho_{h,a,o}}{\sum_{o' \in O} \rho_{h,a,o'}} \sum_{s' \in \mathcal{P}(hao)} \mu_{s,hao,0}. \end{aligned} \tag{35}$$

At each iteration, the **D^{2}NG-POMCP** function applies Thompson sampling to recursively select actions to be executed by simulation from the root node to leaf nodes through the existing search tree T. It inserts each newly visited node into the tree, plays a default rollout policy from the new node, and propagates the simulated outcome to update the hyper-parameters for visited histories, states and actions. The **OnlinePlanning** function is called with current history h and a search tree T initially empty. It repeatedly samples a state from the current belief $\mathcal{P}(h)$, and performs the search process by calling the **D^{2}NG-POMCP** function until some resource budgets are reached (e.g., the computation times out or the maximal number of iterations is reached). Then a greedy action to be performed in the environment is returned to the agent. The **Agent** function is the overall procedure interacting with the real environment. It calls **OnlinePlanning** to select the planned best action, execute the action, get an observation, and update particles by calling the **ParticleFilter** function repeatedly until some terminating conditions are satisfied (e.g., the problem is solved or the maximal running time is reached).

6 Discussion

In this section, we discuss on the advantage of using a simulator, how to choose the prior distributions by initializing hyper-parameters in DNG-MCTS and D^{2}NG-POMCP, and the convergence property of the algorithms.

Fig. 2 Dirichlet-Dirichlet-NormalGamma based partially observable Monte Carlo planing

```
Agent(b_0 : initial belief)
Initialize H ← maximal planning horizon
Initialize 𝓘 ← {possible immediate rewards}
Initialize h ← ∅
Initialize 𝒫(h) ← b_0
repeat
    a ← OnlinePlanning(h, ∅)
    Execute a and get observation o
    h ← hao
    𝒫(h) ← ParticleFilter(𝒫(h), a, o)
until terminating conditions

D²NG-POMCP(s : state, h : history, T : tree, d : depth)
if d ≥ H or s is terminal then
    return 0
else if node ⟨h⟩ is not in tree T then
    Initialize (μ_{s,h,0}, λ_{s,h}, α_{s,h}, β_{s,h}) for s ∈ S, and ρ_{h,a} and ψ_{h,a} for a ∈ A
    Add node ⟨h⟩ to T
    Play rollout policy for H − d steps
    Get cumulative reward r
    return r
else
    a ← ThompsonSampling(h, d, True)
    Execute a by simulation
    Get state s′, observation o and reward i
    h′ ← hao
    𝒫(h′) ← 𝒫(h′) ∪ s′
    r ← i + γ D²NG-POMCP(s′, h′, T, d + 1)
    α_{s,h} ← α_{s,h} + 0.5
    β_{s,h} ← β_{s,h} + (λ_{s,h}(r − μ_{s,h,0})²/(λ_{s,h} + 1))/2
    μ_{s,h,0} ← (λ_{s,h}μ_{s,h,0} + r)/(λ_{s,h} + 1)
    λ_{s,h} ← λ_{s,h} + 1
    ρ_{h,a,o} ← ρ_{h,a,o} + 1
    ψ_{h,a,i} ← ψ_{h,a,i} + 1
    return r

ThompsonSampling(h : history, d : depth, sampling : boolean)
foreach a ∈ A do
    q_a ← QValue(h, a, d, sampling)
return argmax_a q_a

OnlinePlanning(h : history, T : tree)
repeat
    Sample s ∼ 𝒫(h)
    D²NG-POMCP(s, h, T, 0)
until resource budgets reached
return ThompsonSampling(h, 0, False)

QValue(h : history, a : action, d : depth, sampling : boolean)
r ← 0
foreach o ∈ O do
    if sampling = True then
        Sample w_o ∼ Dirichlet(ρ_{h,a})
    else
        w_o ← ρ_{h,a,o} / Σ_{o′∈O} ρ_{h,a,o′}
    h′ ← hao
    r ← r + w_o Value(h′, d + 1, sampling)
r ← γr
foreach i ∈ 𝓘 do
    if sampling = True then
        Sample w_i ∼ Dirichlet(ψ_{h,a})
    else
        w_i ← ψ_{h,a,i} / Σ_{i′∈𝓘} ψ_{h,a,i′}
    r ← r + w_i i
return r

Value(h : history, d : depth, sampling : boolean)
if d ≥ H then
    return 0
else
    if sampling = True then
        Sample (μ_s, τ_s) ∼ NormalGamma(μ_{s,h,0}, λ_{s,h}, α_{s,h}, β_{s,h})
        for s ∈ 𝒫(h)
        return (1/|𝒫(h)|) Σ_{s∈𝒫(h)} μ_s
    else
        return (1/|𝒫(h)|) Σ_{s∈𝒫(h)} μ_{s,h,0}
```

6.1 Use of simulator

The advantage of using a simulator is that a simulator is much more easy to implement and scale than an explicit model of the environment which has to be described as a table of precise probability density/mass functions for each state (i.e. the full transition and/or observation functions in analytic forms). For example, suppose in an idealized environment the state can be described by a scalar variable $x \in (-\infty, \infty)$. After performing an action $a \in [0, 1]$, which is another scalar variable, the next state x' is uncertain, but follows a distribution such that $x' \sim Uniform(x - Uniform(a - Uniform(0, 1), a + Uniform(0, 1)), x + Uniform(a - Uniform(0, 1), a + Uniform(0, 1))$, governed by the environment's own dynamics, where $Uniform(a, b)$ stands for a uniform distribution over range $[a, b]$. Following the MDP formalization, the explicit transition function of this environment in the form of a probability density function is $f(x') = T(x' \mid x, a)$. The distribution $f(x')$ seems to be trivial, but is actually hard to obtain, as it relates to several layers of nested samplings. On the other hand, it is rather easy to implement/describe the environment as a simulator. What we need are the rules of sampling x'

conditioned on x and a — which we already have. It is easy to see that in more complex environments, as shown in this paper, the advantage of using simulator instead of fully specified explicit representations becomes more clear. As the environment becomes complex when more state and action variables are involved, it is extremely difficult or even impossible to maintain the environment's dynamics as explicit distribution functions. But we can always describe the environment as a set of sampling rules — which significantly ease the work of modeling the environment. Furthermore, the simulator doesn't have to be extremely precise. An approximated model of the environment in terms of prior probabilities will suffice. The agent (either modeled as an MDP agent or a POMDP agent) will operate in the abstracted state space inducted by the simulator and keep updating as up-to-date information becomes available. In the case of MDP, the agent act optimally consistent with the simulator assuming that the abstracted/approximated environment model exposed by the simulator is the ground-truth environment; in the case of POMDP, the agent performs real-time Bayesian update conditioned on some prior probabilities and history of observations (in various forms, e.g. particle filtering as in this paper).

6.2 Prior distribution

While the impact of the prior tends to be negligible in the limit, its choice is important especially when only a small amount of data has been observed. In general, priors should reflect available knowledge of the hidden model.

In the absence of any knowledge, *uninformative priors* may be preferred [44]. According to the principle of indifference, uninformative priors assign equal probabilities to all possibilities. For NormalGamma priors, we hope that the sampled distribution of μ given τ: $\mathcal{N}(\mu_0, 1/(\lambda\tau))$, is as flat as possible. This implies an infinite variance $1/(\lambda\tau) \rightarrow \infty$, so that $\lambda\tau \rightarrow 0$. Recall that τ follows a Gamma distribution $Gamma(\alpha, \beta)$ with expectation $\mathbb{E}[\tau] = \alpha/\beta$, so we have expectedly $\lambda\alpha/\beta \rightarrow 0$. Taking into consideration the parameter space ($\lambda > 0, \alpha \geq 1, \beta \geq 0$), we can choose λ small enough, $\alpha = 1$ and β sufficiently large to approximate this condition. Second, we hope the sampled distribution is in the middle of axis, so $\mu_0 = 0$ seems to be a good selection. It is worth noting that intuitively β should not be set too large, or the convergence process may be very slow. For Dirichlet priors, it is common to set the hyper-parameters as small positives to have uninformative priors.

On the other hand, if some prior knowledge is available, *informative priors* make more sense. Take DNG-MCTS as an example. By exploiting domain knowledge, a state node can be initialized with informative priors indicating its priority over other states. In DNG-MCTS, this is done by initializing the hyper-parameters based on subjective estimation of states. For NormalGamma priors, the interpretation of hyper-parameters in terms of pseudo-observations says that if one has a prior mean of μ_0 from λ samples and a prior precision of α/β from 2α samples, the prior distribution over μ and τ is $NormalGamma(\mu_0, \lambda, \alpha, \beta)$ [28], providing a straightforward way to initialize the hyper-parameters if some prior knowledge (such as historical data of past observations) is available. Specifying detailed priors based on prior knowledge for particular domains is beyond the scope of this paper. The ability to include prior information provides important flexibility and can be considered an advantage of the approach.

6.3 Convergence

For Thompson sampling in stationary MABs (i.e., the underlying reward function does not change), Agrawal et al. [2] have shown that: 1) the probability of selecting any suboptimal action a at the current step is bounded by a linear function of the probability of selecting the optimal action; 2) the coefficient in this linear function decreases exponentially fast with the increase in the number of selections of optimal action. Thus, the probability of selecting the optimal action in an MAB is guaranteed to converge to 1 in the limit using Thompson sampling.

Take DNG-MCTS as an example. In our settings, the distribution of $X_{s,\pi}$ is determined by the transition function and the Q values given the policy π. When the Q values converge, the distribution of $X_{s,\pi}$ becomes stationary with the optimal policy. For the leaf nodes (level H) of the search tree, Thompson sampling will converge to the optimal actions with probability 1 in the limit since the MABs are stationary in terms of the default rollout policy. When all the leaf nodes converge, the distributions of return values from them will not change. So the MABs of the nodes in level $H-1$ become stationary as well. Thus, Thompson sampling will also converge to the optimal actions for nodes in level $H-1$. Recursively, this holds for all the upper-level nodes. Therefore, we conclude that DNG-MCTS can find the optimal policy in the search tree for the root node with respect to the default rollout policy if unbounded computational resources are given. We have similar results for D^{2}NG-POMCP. To conclude, DNG-MCTS and D^{2}NG-POMCP converge to find the optimal policy given maximal planning horizon H and the default rollout policies.

7 Experiments

We empirically illustrate our motivation by comparing Thompson sampling with other common algorithms in

MABs. We have also evaluated DNG-MCTS and D^{2}NG-POMCP on various benchmark problems.

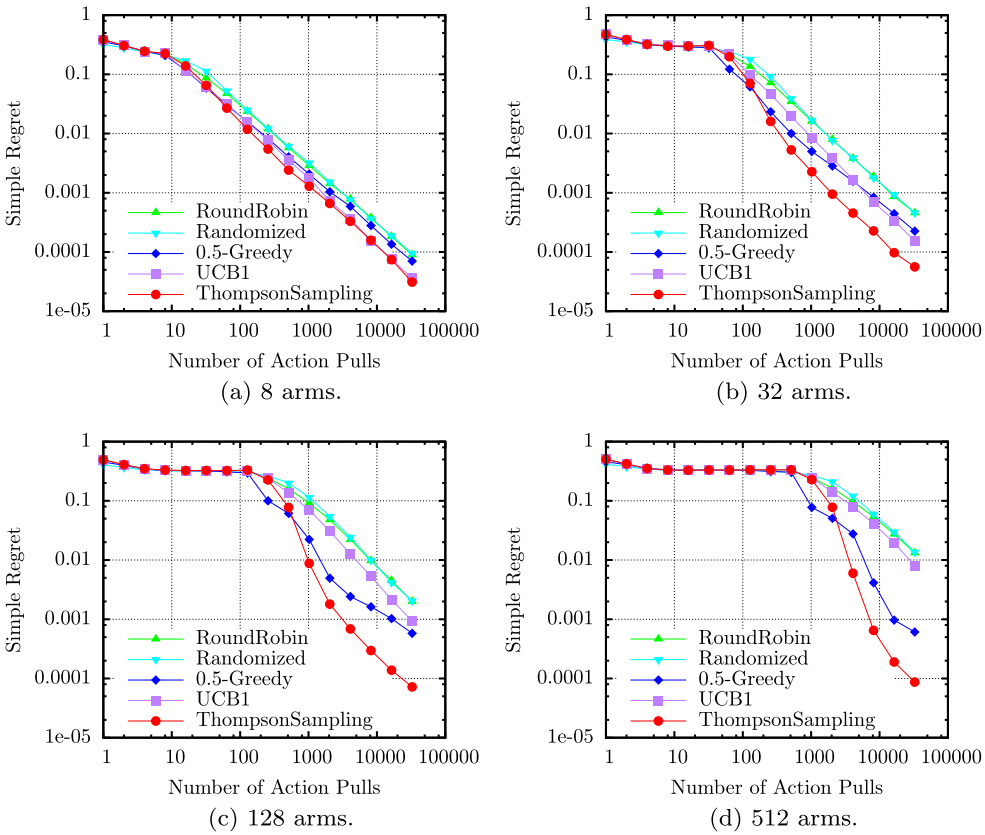
7.1 MAB experiments

We compared Thompson sampling in terms of simple regret with other common algorithms in MABs, including RoundRobin, Randomized, 0.5-Greedy and UCB1. The RoundRobin algorithm selects an arm in a round-robin fashion among all arms [31]; the Randomized algorithm uniformly selects a random arm; the 0.5-Greedy algorithm selects the best seen arm with probability 0.5, and a random arm otherwise [77]; the UCB1 algorithm selects the arm that maximizes the UCB1 heuristic. In our experiments, each arm returns a random reward sampled from a Bernoulli distribution; UCB1 is implemented with exploration constant $\sqrt{2}$; Thompson sampling chooses Beta distributions as the conjugate priors, initialized as $(\alpha = 1, \beta = 1)$. We ran each algorithm over 10,000 experiments of randomly generated instances for different numbers of arms, and reported the average simple regret as shown in Fig. 3. It can be seen from the results that Thompson sampling yields lower simple regret, particularly for larger action space. It is well known that Thompson sampling theoretically achieves logarithmic optimal and empirically performs well in terms of cumulative regret in MABs. To the best of our knowledge, it is observed for the first time in the literature that Thompson sampling empirically outperforms other common algorithms in terms of simple regret also, providing a potential of success of applying Thompson sampling to Monte Carlo planing domains.

7.2 MDP experiments

We have tested the DNG-MCTS algorithm and compared the results with UCT in three common MDP benchmark domains, namely *Canadian traveler problem*, *racetrack* and *sailing*.

These problems are modeled as cost-based MDPs. That is, a cost function $c(s, a)$ is used instead of the reward function $R(s, a)$, and the min operator is used in the Bellman equation instead of the max operator. Similarly, the objective of solving a cost-based MDPs is to find an optimal policy that minimizes the expected cumulative cost for each state. Notice that algorithms developed for reward-based MDPs can be straightforwardly transformed and applied to cost-based MDPs by simply using the min operator instead of max in the Bellman update routines. Accordingly, the min operator is used in the function `ThompsonSampling` of the transformed DNG-MCTS algorithm. We implemented our codes and conducted the experiments on the basis of



(a) 8 arms. (b) 32 arms. (c) 128 arms. (d) 512 arms.

Fig. 3 Performance of Thompson sampling in terms of simple regret in MABs

Table 1 CTP problems with 20 nodes

		Domain-specific UCT		Random rollout policy		Optimistic rollout policy	
Prob.	Belief	UCTB	UCTO	UCT	DNG	UCT	DNG
20-1	20×3^{49}	210.7±7	**169.0±6**	216.4±3	223.9±4	180.7±3	177.1±3
20-2	20×3^{49}	176.4±4	**148.9±3**	178.5±2	178.1±2	160.8±2	155.2±2
20-3	20×3^{51}	150.7±7	**132.5±6**	169.7±4	159.5±4	144.3±3	140.1±3
20-4	20×3^{49}	264.8±9	**235.2±7**	264.1±4	266.8±4	238.3±3	242.7±4
20-5	20×3^{52}	123.2±7	**111.3±5**	139.8±4	133.4±4	123.9±3	122.1±3
20-6	20×3^{49}	165.4±6	**133.1±3**	178.0±3	169.8±3	167.8±2	141.9±2
20-7	20×3^{50}	191.6±6	**148.2±4**	211.8±3	214.9±4	174.1±2	166.1±3
20-8	20×3^{51}	160.1±7	**134.5±5**	218.5±4	202.3±4	152.3±3	151.4±3
20-9	20×3^{50}	235.2±6	**173.9±4**	251.9±3	246.0±3	185.2±2	180.4±2
20-10	20×3^{49}	180.8±7	**167.0±5**	185.7±3	188.9±4	178.5±3	170.5±3
Total		1858.9	**1553.6**	2014.4	1983.68	1705.9	1647.4

The second column indicates the belief size of the transformed MDP for each problem instance. UCTB and UCTO are the two domain-specific UCT implementations [30]. DNG-MCTS and UCT run for 10,000 iterations. Boldface fonts are best in whole table; gray cells show best among domain-independent implementations for each group. The data of UCTB, UCTO and UCT are taken form [19]

`MDP-engine` — a software package with a collection of problem instances and basic algorithms for MDPs.[4]

In each benchmark problem, we (1) ran the transformed algorithms for a number of iterations from the current state, (2) applied the best action based on the resulting action-values, (3) repeated the loop until terminating conditions (e.g., a goal state is satisfied or the maximal number of running steps is reached), and (4) reported the total discounted cost. The performance of algorithms is evaluated by the average value of total discounted costs over 1,000 independent runs. In all experiments, $(\mu_{s,0}, \lambda_s, \alpha_s, \beta_s)$ is initialized to $(0, 0.01, 1, 100)$, and $\rho_{s,a,s'}$ is initialized to 0.01 for all $s \in S$, $a \in A$ and $s' \in S$. For fair comparisons, we also use the same settings as in [19]: for each decision node, (1) only applicable actions are selected, (2) applicable actions are forced to be selected once before any of them are selected twice or more, and 3) the exploration constant for the UCT algorithm is set to be the current mean action-values $Q(s, a, d)$.

The Canadian traveler problem (CTP) is a path finding problem with imperfect information over a graph whose edges may be blocked with given prior probabilities [60]. A CTP can be modeled as a deterministic POMDP, i.e., the only source of uncertainty is the initial belief. When transformed to an MDP, the size of the belief space is $n \times 3^m$, where n is the number of nodes and m is the number of edges. This problem has a discount factor $\gamma = 1$. The aim is to navigate to the goal state as quickly as possible. It has recently been addressed by an anytime variation of AO*, named AOT [19], and two domain-specific implementations of UCT which take advantage of the specific MDP structure of the CTP and use a more informed base policy, named UCTB and UCTO [30]. In this experiment, we used the same 10 problem instances with 20 nodes as done in their papers.

When running DNG-MCTS and UCT in those CTP instances, the number of iterations for each decision-making was set to be 10,000, which is identical to [19]. Two types of default rollout policy were tested: the *random policy* that selects actions with equal probabilities and the *optimistic policy* that assumes traversability for unknown edges and selects actions according to estimated cost. The results are shown in Table 1. Similar to [19], we included the results of UCTB and UCTO as a reference. From the table, we can see that DNG-MCTS outperformed the domain-independent version of UCT with random rollout policy in several instances, and particularly performed much better than UCT with optimistic rollout policy. Although DNG-MCTS is not as good as domain-specific UCTO, it is competitive comparing to the general UCT algorithm in this domain.

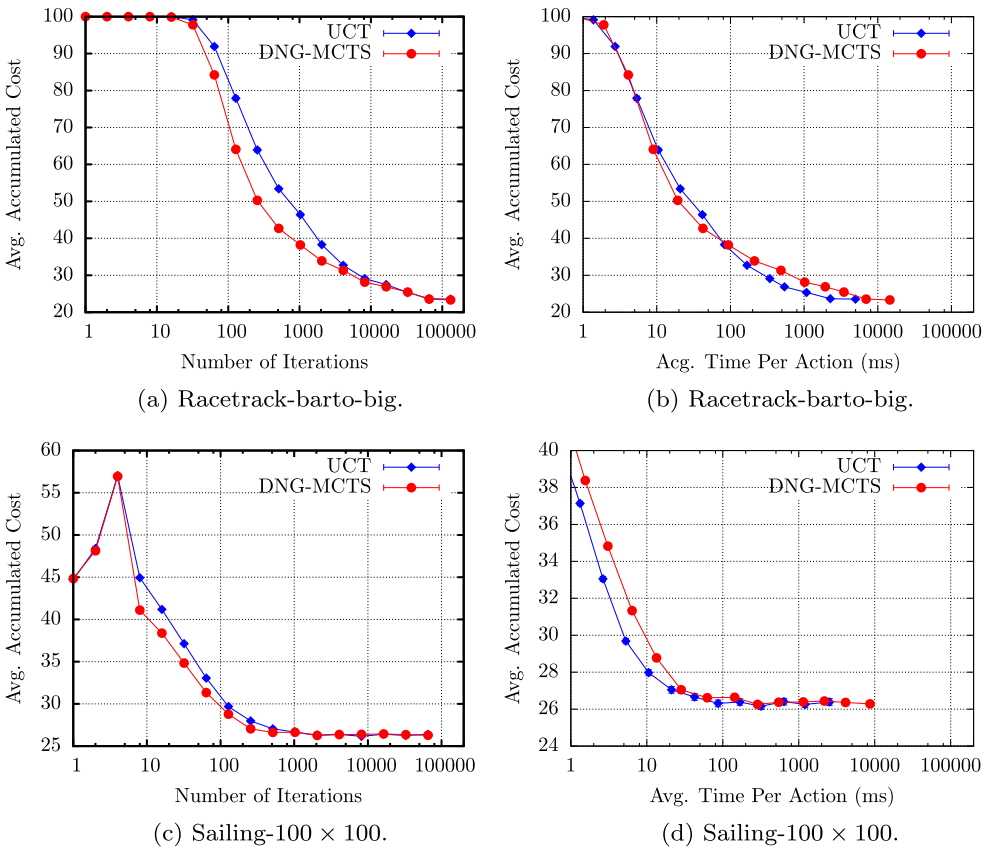
The racetrack problem simulates a car race [15], where a car starts in a set of initial states and moves towards the goal. At each time step, the car can choose to accelerate to one of the eight directions. When moving, the car has a possibility of 0.9 to succeed and 0.1 to fail on its acceleration. We tested DNG-MCTS and UCT with random rollout policy and planning horizon $H = 100$ in the instance of *barto-big*, which has a state space with size $|s| = 22534$. The discount factor is $\gamma = 0.95$ and the optimal cost produced

[4] https://code.google.com/p/mdp-engine/

is known to be 21.38. We report the curves of the average cost as a function of the number of iterations and average computation time per action respectively in Fig. 4a and b. Each data point in the figure was averaged over 1,000 runs, each of which was allowed for running at most 100 steps. It can be seen from the figure that DNG-MCTS converges faster than UCT in terms of number of iterations, and achieves similar results in terms of average computation time per action.

The sailing domain is adopted from [52]. In this domain, a sailboat navigates to a destination on an 8-connected grid. The direction of the wind changes over time according to prior transition probabilities. The goal is to reach the destination as quickly as possible, by choosing at each grid location a neighbour location to move to. The discount factor in this domain is $\gamma = 0.95$ and the maximum planning horizon is set to be $H = 100$. We ran DNG-MCTS and UCT with random rollout policy in a 100×100 instance of this domain. This instance has 80,000 states and the optimal cost is 26.08. The experimental results are shown in Figs. 4c and d. A trend similar to the racetrack problem can be observed in the graph: DNG-MCTS converges faster than UCT in terms of sample complexity.

We also develop an extended Taxi domain to test our algorithm more thoroughly. The original Taxi domain is a common benchmark problem for MDPs [29]. It consists of a 5×5 grid world with walls and 4 taxi terminals: R, G, Y and B. The goal of a taxi agent is to pick up and deliver a passenger. The system has 4 state variables: the agent's coordination x and y, the pickup location pl, and the destination dl. The variable pl can be one of the 4 terminals, or just *taxi* if the passenger is inside the taxi. The variable dl must be one of the 4 terminals. At the beginning of each episode, the taxi's location, the passenger's location and the passenger's destination are all randomly generated. The problem terminates when the taxi agent successfully delivers a passenger. There are 6 primitive actions: a) 4 navigation actions that move the agent one grid: North, South, East and West; b) the Pickup action; and c) the Putdown action. Each navigation action has the probability of 0.8 to move the agent in the indicated direction, and 0.1 for each perpendicular direction. Each action has a cost of -1, and the agent received a reward of +20 when the episode terminates with the Putdown action and a penalty of -10 for illegal Pickup and Putdown actions. In the extended version with size n (i.e., eTaxi[n]), the grid world size is $n \times n$. The four stops R, G, Y and B, are at positions $(0, 0)$, $(0, n-1)$, $(n-2, 0)$ and $(n-1, n-1)$ respectively. There are three walls each with length $\lfloor \frac{n-1}{2} \rfloor$ started at position in between $(0, 0)$ and $(1, 0)$, $(1, n-1)$ and $(2, n-1)$, and $(n-3, 0)$ and $(n-2, 0)$ respectively. The action space, and transition and reward models remain the same as the Taxi



(a) Racetrack-barto-big. (b) Racetrack-barto-big.

(c) Sailing-100 × 100. (d) Sailing-100 × 100.

Fig. 4 Performance comparison on Racetrack and Sailing

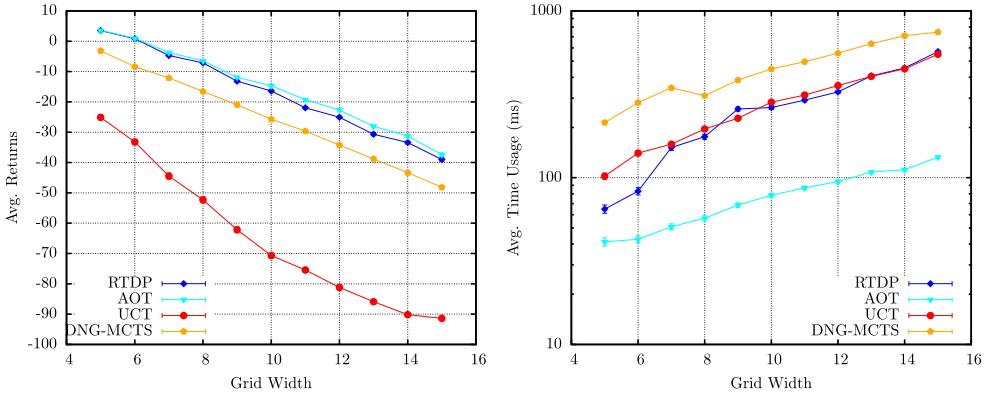


Fig. 5 Performance comparison on eTaxi domain

(a) Average Return. (b) Average Time Usage.

domain. Thus, if $n = 5$, eTaxi[n] reduces to the original Taxi problem.

We compared DNG-MCTS with LRTDP [18], AOT [19] and UCT. In the field of online planning for MDPs, the *real-time dynamic programming* (RTDP) algorithm [15] is among the first that tries to find the "best" action for the current state by conducting a trial-based search process with greedy action selection and an admissible heuristic. The *labeled RTDP* (LRTDP) algorithm introduces a labelling scheme into RTDP that speeds up its convergence while retaining its good anytime behavior. AO* [43] builds an optimal solution graph with respect to the AND-OR graph by greedily expanding tip nodes in the current best partial solution graph and assigning values to new nodes according to an admissible heuristic function. The *anytime AO** (AOT) algorithm implements AO* in an anytime manner retaining the same optimality and worst case complexity as AO*. Notice that, LRTDP and AOT are not in Monte Carlo settings, that is to say they have full accesses to the underlying MDP models. A *min-min* heuristic [18] is used to initialize new nodes in LRTDP and AOT, and a *min-min* heuristic based greedy policy is used as the base policy for UCT and DNG-MCTS. In the experiments, we ran each algorithm with randomly selected initial states, and reported the return (cumulative reward) and the overall time usage. The number of iterations for each action-selection is set to be 100. The maximal search depth is 100. We conducted the experiments over different sizes of eTaxi ranging from $n = 5$ to 15. The average returns and time usages over 1,000 runs are reported in Fig. 5a and b. Detailed results in eTaxi[5] is shown in Table 2. It can be seen from the results that DNG-MCTS performs much better than UCT, while producing competitive results comparing to LRTDP and AOT with regard to average cumulative rewards in eTaxi domain. We also notice that DNG-MCTS does require more computation time comparing to other algorithms, this is due to the time consuming operation of sampling from variety of distributions.

Table 2 Empirical results in eTaxi[5]

Algorithm	Runs	Avg. Reward	Avg. Time usage (ms)
LRTDP	1000	3.71 ± 0.15	64.88 ± 3.71
AOT	1000	3.80 ± 0.16	41.26 ± 2.37
UCT	1000	-23.10 ± 0.84	102.20 ± 4.24
DNG-MCTS	1000	-3.13 ± 0.29	213.85 ± 4.75

The upper bound of Average Reward is 4.01 ± 0.15

7.3 POMDP experiments

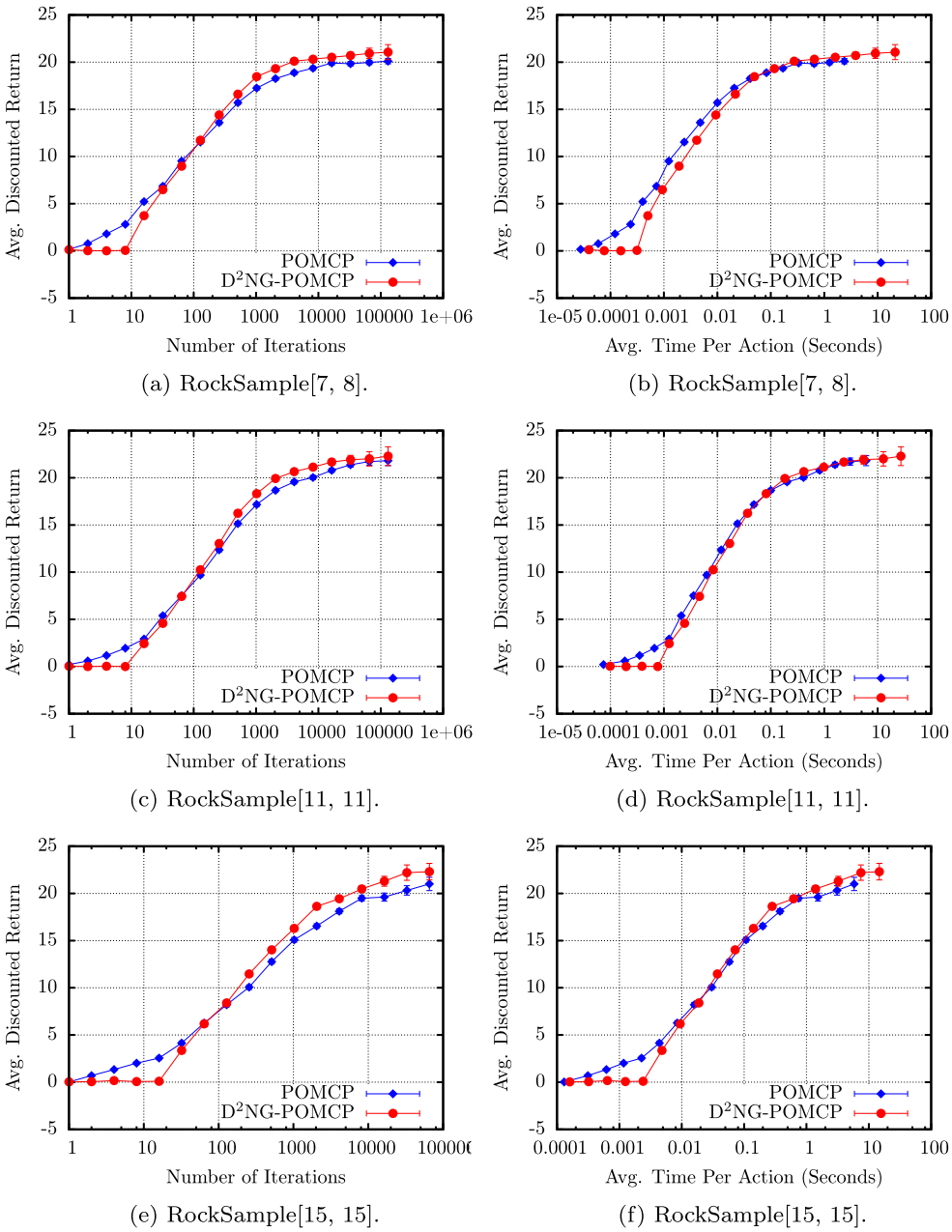
We have also examined D^{2}NG-POMCP and compared the results with POMCP in *RockSample* and *PocMan* domains. We implemented the algorithm and conducted the experiments based on `POMCP` — a software package implementing the POMCP algorithm with some benchmark problems.[5]

For each problem instance, we 1) ran the algorithms for a number of iterations from the current history, 2) applied the best action based on the resulting action-values, 3) repeated the loop until terminating conditions, and 4) reported the total discounted reward and the average computation time per action selected. The performance of algorithms is evaluated by the average total discounted reward over 1,000 independent runs. In all experiments, we initialized $(\mu_{s,h,0}, \lambda_{s,h}, \alpha_{s,h}, \beta_{s,h})$ to $(0, 0.01, 1, 100)$, $\psi_{h,a,r}$ to 0.01, and $\rho_{h,a,o}$ to 0.01 for all $s \in S$, $a \in A$, $r \in \mathcal{I}$, $o \in O$ and encountered history h in the search tree. When testing D^{2}NG-POMCP and POMCP, we used the same preferred actions based rollout policy as described and implemented in [70] and `POMCP` respectively. For fair comparisons, we also applied the same settings as in `POMCP`: for each decision node, 1) only applicable actions are selected, and

[5] http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Applications.html

2) applicable actions are forced to be selected once before any of them is selected twice.

The RockSample[n, k] problem simulates a robot exploring in a $n \times n$ grid map containing k rocks. The goal is to determine which rocks are valuable, collect valuable ones as much as possible and exit from the map finally. There are three kinds of actions, namely *moving*, *checking* and *sampling*. The *moving* action specified with a direction parameter enables the agent to move in the map; the *sampling* action collects a rock; and, the *checking* action observes a rock to noisily identify whether it is valuable or not. Sampling a valuable rock yields a reward of $+10$; sampling an invaluable rock yields a reward of -10; and, moving into the exit area yields a reward of $+10$. All other actions have no cost or reward. The discount factor γ is 0.95. Empirical results are depicted in Fig. 6. Each data point shows the average result over 1,000 runs or at most 12 hours of total computation. The three figures in the left column depict the performance with respect to number of iterations; the three figures in the right column depict the performance with respect to average time per action. Number of iterations is the total number of calls to **D^{2}NG-POMCP** allowed for each decision of action in the **OnlinePlanning** function; average time per action is the average computation time for each action selected in the **OnlinePlanning** function. We can see from the results that D^{2}NG-POMCP converges



(a) RockSample[7, 8]. (b) RockSample[7, 8].

(c) RockSample[11, 11]. (d) RockSample[11, 11].

(e) RockSample[15, 15]. (f) RockSample[15, 15].

Fig. 6 Performance comparison on RockSample

Table 3 Comparison of D^{2}NG-POMCP with existing approaches in RockSample problems evaluated by discounted cumulative reward

RockSample	[7, 8]	[11,11]	[15,15]
States $\|s\|$	12,544	247,808	7,372,800
AEMS2	21.37 ± 0.22	N/A	N/A
HSVI-BFS	21.46 ± 0.22	N/A	N/A
SARSOP	21.39 ± 0.01	21.56 ± 0.11	N/A
POMCP	20.71 ± 0.21	20.01 ± 0.23	15.32 ± 0.28
D^{2}NG-POMCP	20.87 ± 0.20	21.44 ± 0.21	20.20 ± 0.24

faster than POMCP in terms of number of iterations, and appears to be competitive with POMCP in terms of average time per action.

Table 3 presents the comparison of D^{2}NG-POMCP with prior work, including AEMS2 [65], HSVI-BFS [66, 71], SARSOP [54] and POMCP [70]. AEMS2 and HSVI-BFS are online algorithms; SARSOP is an offline algorithm. They are all provided with full factored representations of the underlying problems. AEMS2 and HSVI-BFS used knowledge computed offline by PBVI [63]; empirical results are taken from [66]. SARSOP was given approximately 1,000 seconds of offline computation; results are taken from [54]. POMCP and D^{2}NG-POMCP used the same informed rollout policy. The results of POMCP are taken from [70]. Each online algorithm was given exactly 1 second per action. Performance is evaluated by average discounted return over 1,000 runs or at most 12 hours of total computation. The results indicate that D^{2}NG-POMCP is able to provide competitive results on RockSample[7, 8] and RockSample[11, 11], and advanced POMCP with much better return on RockSample[15, 15].

The PocMan problem is firstly introduced in [70]. The PocMan agent navigates in a 17×19 maze, while trying to eat some randomly distributed food pellets and power pills. Four ghost agents roam the maze, according to a given stochastic strategy. The PocMan agent dies if it touches any ghost, unless it has eaten any power pills within the last 15 time steps. It receives a reward of -1 at each step, $+10$ for each food pellet, $+25$ for eating a ghost and -100 for dying. A 10-bit observation is observed at every time step, corresponding to the PocMan agent's senses of sight, hearing, touch and smell. The PocMan problem has approximately 1,056 states, 4 actions, and 1,024 observations. The discount factor γ is 0.95. The performance of D^{2}NG-POMCP in PocMan evaluated by average discounted return is shown in Fig. 7. Each data point shows the average result over 1,000 runs or at most 12 hours of total computation. It is worth noticing that the algorithm's performance in PocMan experiment is evaluated by average discounted return, instead of average undiscounted return as in the original paper [70]. We believe that using average discounted return to show the performance is more reasonable, since average discounted return is what the algorithms really intent to optimize. To provide a more comprehensive view, we also included the respective results evaluated by average undiscounted return in addition as shown in Fig. 8. In this domain, D^{2}NG-POMCP performs much better than POMCP with regard to both number of iterations and average time per action.

7.4 Discussion on computational complexity

Regarding computational complexity, although the total computation time of DNG-MCTS and D^{2}NG-POMCP is linear with the total number of simulations, which is at most $width \times depth$ (where $width$ is the number of iterations and $depth$ is the maximal planning horizon), our approaches do require more computation than UCT and POMCP, due to the time consuming operations of samplings from various distributions when performing Thompson sampling. However, if the simulations are expensive (e.g., computational physics in 3D environment or stochastic environment with multiple agents where the cost of executing the simulation steps greatly exceeds the time needed by action-selection steps in MCTS), DNG-MCTS and D^{2}NG-POMCP can obtain better performances in terms of computational complexity, because they are

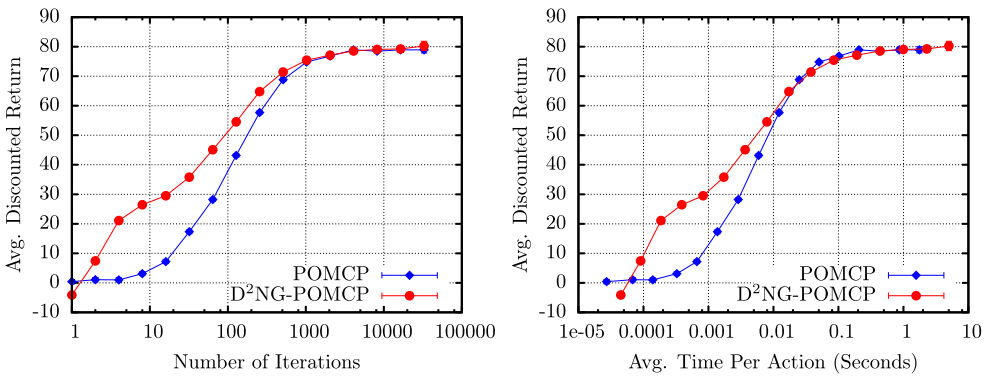


Fig. 7 Performance of D^{2}NG-POMCP in PocMan evaluated by average discounted returns

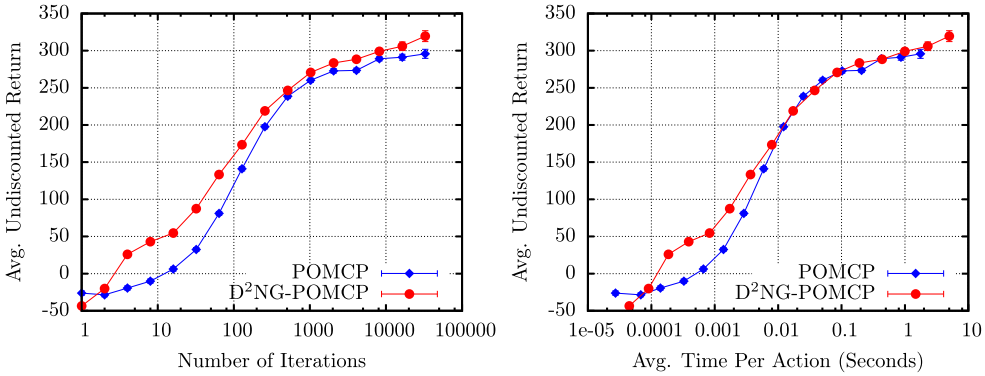


Fig. 8 Performance of D^{2}NG-POMCP in PocMan evaluated by average undiscounted returns

expected to have lower sample complexities (for example, as confirmed in the PocMan experiments).

8 Conclusion

In this paper, we propose the DNG-MCTS and D^{2}NG-POMCP algorithms which reply on Thompson sampling for online Monte Carlo planning for MDPs and POMDPs. The basic idea is to model the uncertainty of the cumulative reward returned by taking an action in the Monte Carlo search tree as a combination of mixture distributions, infer the posterior distribution using Bayesian method, and use Thomson sampling to guide the action-selection strategy. We show that the proposed algorithms are guaranteed to converge to the optimal policy in the limit. Experimental results in MDPs confirm that, comparing with the general UCT algorithm, DNG-MCTS produces competitive results in the CTP domain, and converges faster in the domains of racetrack and sailing. We also show that DNG-MCTS outperforms significantly other popular online planning algorithms (including RTDP, AOT and UCT) on the eTaxi domain which requires complex behaviors. Experimental results in POMDPs show that D^{2}NG-POMCP outperform the state-of-the-art algorithms (including AEMS2, HSVI-BFS, SARSOP, and POMCP) in both RockSample and PocMan domains. In future work, we plan to extend our basic assumptions to more general distributions and test our algorithm on real-world applications.

Acknowledgements Feng Wu was supported in part by National Natural Science Foundation of China under grant No. 61603368, the Youth Innovation Promotion Association of CAS (No. 2015373), and Natural Science Foundation of Anhui Province under grant No. 1608085QF134. Aijun Bai was supported in part by the National Research Foundation for the Doctoral Program of China under grant 20133402110026, the National Hi-Tech Project of China under grant 2008AA01Z150 and the Natural Science Foundation of China under grant 60745002 and 61175057. We are grateful to the reviewers for their constructive comments and suggestions.

References

1. Agrawal S, Goyal N (2012) Analysis of thompson sampling for the multi-armed bandit problem. In: Conference on learning theory, pp 39.1–39.26
2. Agrawal S, Goyal N (2013) Further optimal regret bounds for Thompson sampling. In: Artificial intelligence and statistics, pp 99–107
3. Anand A, Mausam GA, Singla P (2015) ASAP-UCT: Abstraction of state-action pairs in UCT. In: Yang Q, Wooldridge M (eds) IJCAI. AAAI Press, pp 1509–1515
4. Anand A, Mausam RN, Singla P (2016) OGA-UCT: On-the-go abstractions in UCT. In: Coles AJ, Coles A, Edelkamp S, Magazzeni D, Sanner S (eds) ICAPS. AAAI Press, pp 29–37
5. Asmuth J, Littman ML (2011) Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search. In: Uncertainty in artificial intelligence, pp 19–26
6. Auer P (2003) Using confidence bounds for exploitation-exploration trade-offs. J Mach Learn Res 3:397–422
7. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn 47(2):235–256
8. Bai A, Srivastava S, Russell S (2016) Markovian state and action abstractions for MDPs via hierarchical MCTS. In: 25th international joint conference on artificial intelligence (IJCAI). New York
9. Bai A, Wu F, Chen X (2012) Online planning for large MDPs with MAXQ decomposition (extended abstract). In: van der Hoek W, Padgham L, Conitzer V, Winikoff M (eds) International conference on autonomous agents and multiagent systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 volumes). IFAAMAS, pp 1215–1216
10. Bai A, Wu F, Chen X (2013) Bayesian Mixture modelling and inference based Thompson sampling in Monte-Carlo tree search. In: Advances in neural information processing systems 26, pp 1646–1654
11. Bai A, Wu F, Chen X (2015) Online planning for large Markov decision processes with hierarchical decomposition. ACM Trans Intell Syst Technol (TIST) 6(4):45:1–45:28
12. Bai A, Wu F, Zhang Z, Chen X (2014) Thompson sampling based Monte-Carlo planning in POMDPs. In: International conference on automated planning and scheduling (ICAPS)
13. Barrett S, Agmon N, Hazon N, Kraus S, teammates P. Stone. (2014) Communicating with unknown. In: Proceedings of 13th international conference on autonomous agents and multiagent systems (AAMAS 2012)

14. Barrett S, Stone P, Kraus S, Rosenfeld A (2013) Teamwork with limited knowledge of teammates. In: Proceedings of the twenty-seventh AAAI conference on artificial intelligence
15. Barto A, Bradtke S, Singh S (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1-2):81–138
16. Bellman R (1957) Dynamic programming, 1st edn. Princeton University Press, Princeton
17. Bertsekas DP, Castanon DA (1999) Rollout algorithms for stochastic scheduling problems. J Heuristics 5(1):89–108
18. Bonet B, Geffner H (2003) Labeled rtdp: Improving the convergence of real-time dynamic programming. In: International conference on automated planning and scheduling, vol 3
19. Bonet B, Geffner H (2012) Action selection for MDPs Anytime AO* vs. UCT. In: AAAI conference on artificial intelligence, pp 1749–1755
20. Browne C, Powley EJ, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo, tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43
21. Bubeck S, Cesa-Bianchi N (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found Trends Mach Learn 5(1):1–122
22. Bubeck S, Munos R, Stoltz G (2011) Pure exploration in finitely-armed and continuous-armed bandits. Theor Comput Sci 412(19):1832–1852
23. Chang HS, Givan R, Chong EK (2004) Parallel rollout for online solution of partially observable Markov decision processes. Discret Event Dyn Syst 14(3):309–341
24. Chapelle O, Li L (2011) An empirical evaluation of Thompson sampling. In: Advances neural information processing systems, pp 2249–2257
25. Chaslot G, Bakkes S, Szita I, Spronck P (2008) Monte-Carlo tree search: a new framework for game AI. In: Darken C, Mateas M (eds) Proceedings of the fourth artificial intelligence and interactive digital entertainment conference. The AAAI Press, Stanford
26. DasGupta A (2008) Asymptotic theory of statistics and probability. Springer, Berlin
27. Dearden R, Friedman N, Russell S (1998) Bayesian Q-learning. In: AAAI conference on artificial intelligence, pp 761–768
28. DeGroot MH, Schervish MJ (2002) Probability and statistics. Addison Wesley, Boston
29. Dietterich TG (1999) Hierarchical reinforcement learning with the MAXQ value function decomposition. J Mach Learn Res 13(1):63
30. Eyerich P, Keller T, Helmert M (2010) High-quality policies for the Canadian traveler's problem. In: AAAI conference on artificial intelligence, pp 51–58
31. Feldman Z, Domshlak C (2012) Simple regret optimization in online planning for Markov decision processes. In: AAAI conference on artificial intelligence
32. Feldman Z, Domshlak C (2014) On MABs and separation of concerns in Monte-Carlo planning for MDPs. In: Chien SA, Do MB, Fern A, Ruml W (eds) ICAPS. AAAI
33. Feng Z, Hansen E (2002) Symbolic heuristic search for factored Markov decision processes. In: AAAI/IAAI, pp 455–460
34. Finnsson H, Björnsson Y (2008) Simulation-based approach to general game playing. AAAI 8:259–264
35. Forbes C, Evans M (2011). In: Hastings N, Peacock B (eds) Statistical distributions. Wiley, Nwe York
36. Gelly S, Silver D (2007) Combining online and offline knowledge in UCT. In: Proceedings of the 24th international conference on machine learning. ACM, pp 273–280
37. Gelly S, Silver D (2011) Monte-Carlo Tree search and rapid action value estimation in computer Go. Artif Intell 175(11):1856–1875
38. Gopalan A, Mannor S, Mansour Y (2014) Thompson sampling for complex online problems. In: Proceedings of the 31st international conference on machine learning, pp 100–108
39. Gordon NJ, Salmond DJ, Smith AF (1993) Novel approach to nonlinear/non-Gaussian bayesian state estimation. In: IEE Proceedings F (radar and signal processing), vol 140. IET, pp 107–113
40. Grzes M, Poupart P (2014) Pomdp planning and execution in an augmented space. In: Proceedings of the 2014 international conference on autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, pp 757–764
41. Grześ M, Poupart P, Hoey J (2013) Isomorph-free branch and bound search for finite state controllers. In: Proceedings of the twenty-third international joint conference on artificial intelligence. AAAI Press, pp 2282–2290
42. Guez A, Silver D, Dayan P (2012) Efficient Bayes-adaptive reinforcement learning using sample-based search. In: Advances in neural information processing systems, pp 1034–1042
43. Hansen E, Zilberstein S (2001) LAO* A heuristic search algorithm that finds solutions with loops. Artif Intell 129(1-2):35–62
44. Jaynes ET (1968) Prior probabilities. IEEE Trans Syst Sci Cybern 4(3):227–241
45. Jones GL (2004) On the Markov chain central limit theorem. Probab Surv 1:299–320
46. Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101(1-2):99–134
47. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
48. Kaufmann E, Korda N, Munos R (2012) Thompson sampling An optimal finite time analysis. In: Algorithmic Learning Theory, pp 199–213
49. Kearns M, Mansour Y, Ng A (1999) A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In: Proceedings of the 16th international joint conference on artificial intelligence, vol 2. Morgan Kaufmann Publishers Inc, pp 1324–1331
50. Keller T, Eyerich P (2012) Prost: Probabilistic planning based on UCT. In: ICAPS12
51. Keller T, Helmert M (2013) Trial-based heuristic tree search for finite horizon MDPs. In: Proceedings of the 23rd international conference on automated planning and scheduling (ICAPS), pp 135–143
52. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: European conference on machine learning, pp 282–293
53. Korda N, Kaufmann E, Munos R (2013) Thompson sampling for 1-dimensional exponential family bandits. In: Burges C, Bottou L, Welling M, Ghahramani Z, Weinberger K (eds) Advances in neural information processing systems 26. Curran Associates, Inc, pp 1448–1456
54. Kurniawati H, Hsu D, Lee WS (2008) SARSOP efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: science and systems, pp 65–72
55. Lai T, Robbins H (1985) Asymptotically efficient adaptive allocation rules. Adv Appl Math 6:4–22
56. Macindoe O, Kaelbling LP, Lozano-Pérez T (2012) POMCoP: Belief space planning for sidekicks in cooperative games. In: Riedl M, Sukthankar G (eds) Proceedings of the eighth AAAI conference on artificial intelligence and interactive digital entertainment, AIIDE-12. The AAAI Press, Stanford
57. McAllester DA, Singh S (1999) Approximate planning for factored pomdps using belief state simplification. In: Proceedings

of the fifteenth conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc, pp 409–416
58. McMahan HB, Likhachev M, Gordon G (2005) Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In: Proceedings of the 22nd international conference on machine learning. ACM, pp 569–576
59. Osband I, Russo D, Van Roy B (2013) (more) efficient reinforcement learning via posterior sampling. In: Advances in neural information processing systems, pp 3003–3011
60. Papadimitriou CH, Yannakakis M (1991) Shortest paths without a map. Theor Comput Sci 84(1):127–150
61. Paquet S, Chaib-draa B, Ross S (2006) Hybrid POMDP Algorithms. In: Proceedings of the workshop on multi-agent sequential decision making in uncertain domains (MSDM-06). Citeseer, pp 133–147
62. Paquet S, Tobin L, Chaib-draa B (2005) Real-time decision making for large POMDPs. In Advances in artificial intelligence. Springer, pp 450–455
63. Pineau J, Gordon G, Thrun S et al (2003) Point-based value iteration: an anytime algorithm for POMDPs. In: IJCAI, vol 3, pp 1025–1032
64. Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York
65. Ross S, Chaib-Draa B et al (2007) Aems: an anytime online search algorithm for approximate policy refinement in large POMDPs. In: IJCAI, pp 2592–2598
66. Ross S, Pineau J, Paquet S, Chaib-Draa B (2008) Online planning algorithms for POMDPs. J Artif Intell Res 32(1):663–704
67. Sanner S, Goetschalckx R, Driessens K, Shani G (2009) Bayesian real-time dynamic programming. In: IJCAI, pp 1784–1789
68. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529(7587):484–489
69. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A et al (2017) Mastering the game of go without human knowledge. Nature 550(7676):354
70. Silver D, Veness J (2010) Monte-Carlo planning in large POMDPs. In: Advances in neural information processing systems, pp 2164–2172
71. Smith T, Simmons R (2004) Heuristic search value iteration for POMDPs. In: Proceedings of the 20th conference on uncertainty in artificial intelligence. AUAI Press, pp 520–527
72. Somani A, Ye N, Hsu D, Lee WS (2013) DESPOT: Online POMDP planning with regularization. In: Burges C, Bottou L, Welling M, Ghahramani Z, Weinberger K (eds) Advances in neural information processing systems 26. Curran Associates, Inc, pp 1772–1780
73. Sutton RS, Barto AG (1998) Reinforcement learning: An introduction. The MIT Press, Cambridge
74. Tesauro G, Rajan VT, Segal R (2010) Bayesian inference in Monte-Carlo tree search. In: Uncertainty in artificial intelligence, pp 580–588
75. Thompson WR (1933) On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25:285–294
76. Thrun S (1999) Monte Carlo POMDPs. In: NIPS, vol 12, pp 1064–1070
77. Tolpin D, Shimony SE (2012) MCTS based on simple regret. In: AAAI conference on artificial intelligence
78. Vien NA, Ertel W, Dang V-H, Chung T (2013) Monte-Carlo tree search for bayesian reinforcement learning. Appl Intell 39(2):345–353
79. Wang T, Lizotte D, Bowling M, Schuurmans D (2005) Bayesian sparse sampling for on-line reward optimization. In: Proceedings of the 22nd international conference on machine learning. ACM, pp 956–963
80. Washington R (1997) BI-POMDP: bounded, incremental partially-observable Markov-model planning. In: Recent advances in AI planning. Springer, pp 440–451
81. Winands MH, Bjornsson Y, Saito J (2010) Monte Carlo tree search in lines of action. IEEE Trans Comput Intell AI Games 2(4):239–250
82. Wu F, Zilberstein S, Chen X (2011) Online planning for ad hoc autonomous agent teams. In: International joint conference on artificial intelligence, pp 439–445
83. Zhang Z, Chen X (2012) FHHOP a factored hybrid heuristic online planning algorithm for large POMDPs. In: Proceedings of the 28th conference on uncertainty in artificial intelligence. Catalina Island, pp 934–943