A Surprisingly Simple Continuous-Action POMDP Solver: Lazy Cross-Entropy Search Over Policy Trees

Marcus Hoerger[1], **Hanna Kurniawati**[2], **Dirk Kroese**[1], **Nan Ye**[1]

[1]School of Mathematics & Physics, The University of Queensland, Queensland, Australia
[2]School of Computing, Australian National University, ACT, Australia
m.hoerger@uq.edu.au, hanna.kurniawati@anu.edu.au, kroese@maths.uq.edu.au, nan.ye@uq.edu.au

Abstract

The Partially Observable Markov Decision Process (POMDP) provides a principled framework for decision making in stochastic partially observable environments. However, computing good solutions for problems with continuous action spaces remains challenging. To ease this challenge, we propose a simple online POMDP solver, called Lazy Cross-Entropy Search Over Policy Trees (LCEOPT). At each planning step, our method uses a novel *lazy* Cross-Entropy method to search the space of policy trees, which provide a simple policy representation. Specifically, we maintain a distribution on promising finite-horizon policy trees. The distribution is iteratively updated by sampling policies, evaluating them via Monte Carlo simulation, and refitting them to the top-performing ones. Our method is lazy in the sense that it exploits the policy tree representation to avoid redundant computations in policy sampling, evaluation, and distribution update. This leads to computational savings of up to two orders of magnitude. Our LCEOPT is surprisingly simple as compared to existing state-of-the-art methods, yet empirically outperforms them on several continuous-action POMDP problems, particularly for problems with higher-dimensional action spaces.

Introduction

Decision making in stochastic partially observable environments is an essential, yet challenging problem in many domains, such as robotics (Kurniawati 2022), natural resource management (Filar, Qiao, and Ye 2019) and cybersecurity (Schwartz, Kurniawati, and El-Mahassni 2020). The Partially Observable Markov Decision Process (POMDP) provides a principled framework to solve such decision making problems, by lifting the planning problem from an agent's state space to its *belief space* i.e., the space of all probability distributions over the state space. While solving POMDPs exactly is computationally intractable in general (Papadimitriou and Tsitsiklis 1987), many efficient approximately-optimal sampling-based online POMDP solvers have been developed (reviewed in Kurniawati (2022)), making them viable tools for many realistic decision making problems under uncertainty.

However, solving POMDPs with continuous and high-dimensional action spaces remains challenging. Current state-of-the-art online solvers for POMDPs with continuous action spaces (Seiler, Kurniawati, and Singh 2015; Sunberg and Kochenderfer 2018; Mern et al. 2021; Lim, Tomlin, and Sunberg 2021; Hoerger et al. 2023) typically use Monte Carlo Tree Search (MCTS) (Coulom 2007) to find a near-optimal action amongst a sampled representative subset of the action space, often relying on partitioning of the action space, diminishing their performance for high-dimensional action spaces.

We propose a new simple online POMDP solver for continuous action spaces, Lazy Cross-Entropy Search Over Policy Trees (LCEOPT), that uses a stochastic optimization approach in the policy space by extending the Cross-Entropy method for optimization (Rubinstein and Kroese 2004; de Boer et al. 2005) to compute a near-optimal policy, while avoiding any form of action-space partitioning. LCEOPT represents a policy as a *policy tree*, a compact and interpretable representation that gives rise to simple policy parameterizations via finite-dimensional vectors. Following the standard procedure of the Cross-Entropy method, LCEOPT maintains a parameterized distribution over the policy parameters that is incrementally updated by sampling sets of parameters from the distribution and evaluating their associated policies via Monte Carlo sampling. The distribution is then updated towards the best-performing policies. This enables LCEOPT to quickly focus its search on promising regions of the policy space.

LCEOPT assumes independence of the marginal distributions over each component of the parameter vectors. This assumption allows us to derive a lazy parameter sampling, evaluation and distribution update method which only samples parts of a policy tree that are relevant for its evaluation. Our lazy approach reduces the cost of sampling policies by up to two orders of magnitude for problems with higher-dimensional action spaces, thereby significantly increasing the overall efficiency of LCEOPT.

In contrast to many MCTS-based solvers, LCEOPT avoids any form of partitioning of the action space, enabling it to scale much more effectively to problems with higher-dimensional action spaces. Despite its simplicity, LCEOPT achieves remarkable results in various benchmark problems with continuous action spaces compared to current state-of-the-art methods, particularly for problems with higher-dimensional action spaces (up to 12-D). The source code

of LCEOPT is available at https://github.com/hoergems/LCEOPT.

Related Work

Various efficient sampling-based online POMDP solvers have been developed for increasingly complex discrete and continuous POMDPs in the last two decades. In contrast to offline methods (Bai, Hsu, and Lee 2014; Kurniawati et al. 2010; Kurniawati, Hsu, and Lee 2008; Pineau, Gordon, and Thrun 2003; Smith and Simmons 2005) that compute a policy offline before deployment, online solvers (e.g., (Kurniawati and Yadav 2016; Silver and Veness 2010; Ye et al. 2017)) aim to further scale to larger and more complex problems by interleaving planning and execution, and focus on computing an optimal action for only the current belief during planning. For scalability purposes, LCEOPT follows the online solving approach.

Some online solvers have been designed for continuous POMDPs, most of them being MCTS-based (Seiler, Kurniawati, and Singh 2015; Sunberg and Kochenderfer 2018; Mern et al. 2021; Lim, Tomlin, and Sunberg 2021; Hoerger et al. 2023) with some relying on partitioning the action space (Lim, Tomlin, and Sunberg 2021; Hoerger et al. 2023). These solvers do not scale well to problems with higher-dimensional action spaces though.

The Cross-Entropy method has been used in several algorithms for solving POMDPs and MDPs (the fully observable variant of POMDPs). Several of them consider discrete action spaces (Mannor, Rubinstein, and Gat 2003; Oliehoek, Kooij, and Vlassis 2008; Wang, Kurniawati, and Kroese 2018), while we consider POMDPs with continuous action spaces. Omidshafiei et al. (2016) consider continuous actions spaces, but the optimization is carried out over a finite policy space. Hafner et al. (2019) presents a Cross-Entropy based POMDP solver within a deep planning framework that optimizes over open-loop policies, while our method optimizes over closed-loop policies.

Additionally, some solvers (Agha-mohammadi, Chakravorty, and Amato 2011; Sun, Patil, and Alterovitz 2015; van den Berg, Abbeel, and Goldberg 2011; van den Berg, Patil, and Alterovitz 2012) restrict beliefs to be Gaussian and use Linear-Quadratic-Gaussian (LQG) control (Lindquist 1973) to compute the best action. This strategy generally performs well in high-dimensional action spaces. However, they tend to perform poorly in problems with large uncertainties or non-Gaussian beliefs (Hoerger et al. 2020). In contrast, our method puts no restriction on the class of beliefs, while simultaneously retaining efficiency in higher-dimensional action spaces.

Preliminaries

Partially Observable Markov Decision Process (POMDP) A POMDP provides a general mathematical framework for sequential decision making under uncertainty. Formally, a POMDP is an 8-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$. Initially, the agent is in a hidden state $s_0 \in \mathcal{S}$. This uncertainty is represented by an initial belief $b_0 \in \mathcal{B}$, a probability distribution on the state space $\mathcal{S}$, where $\mathcal{B}$ is the set of all possible beliefs. At each step $t \geq 0$, the agent executes an action $a_t \in \mathcal{A}$ according to some policy π. Due to stochastic effects of executing actions, it transitions from the current state $s_t \in \mathcal{S}$ to a next state $s_{t+1} \in \mathcal{S}$ according to the transition model $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t)$. For discrete state spaces, $T(s_t, a_t, s_{t+1})$ is often a probability mass function, whereas for continuous state spaces, it typically is a probability density function. The agent does not know the state s_{t+1} exactly, but perceives an observation $o_t \in \mathcal{O}$ from the environment according to the observation model $Z(s_{t+1}, a_t, o_t) = p(o_t|s_{t+1}, a_t)$. In addition, the agent receives an immediate reward $r_t = R(s_t, a_t) \in \mathbb{R}$. The agent's goal is to find a policy π that maximizes the expected total discounted reward or the *policy value*

$$V_\pi(b_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, b_0, \pi\right], \tag{1}$$

where the discount factor $0 < \gamma < 1$ ensures that $V_\pi(b)$ is finite and well-defined.

The agent's decision space is the set Π of policies, defined as mappings from beliefs to actions. The POMDP solution is then the optimal policy, denoted as π^* and given by

$$\pi^* = \arg\max_{\pi \in \Pi} V_\pi(b), \tag{2}$$

for each belief $b \in \mathcal{B}$. A more elaborate explanation is available in Kaelbling, Littman, and Cassandra (1998).

Cross-Entropy Method for Optimization The Cross-Entropy (CE) Method (Rubinstein and Kroese 2004; Botev et al. 2013) is a gradient-free method for discrete and continuous optimization problems. Suppose $\mathcal{X}$ is an arbitrary solution space, and $f : \mathcal{X} \to \mathbb{R}$ is an objective function that we aim to optimize, i.e., we aim to find $x^* \in \mathcal{X}$, such that $x^* = \arg\max_{x \in \mathcal{X}} f(x)$. To do this, the CE-method iteratively constructs a sequence of sampling densities $d(\cdot; \eta_1), d(\cdot; \eta_2), \ldots, d(\cdot; \eta_T)$ over $\mathcal{X}$, with parameters $\eta_1, \ldots, \eta_T$ such that $d(\cdot; \eta_t)$ assigns more probability mass near x^* as t increases.

In particular, suppose we start from an initial sampling density $d(\cdot; \eta_1)$. At iteration $1 \leq t \leq T$, the CE-method draws a sample of candidate solutions $X = \{x_i\}_{i=1}^N$ from $d(\cdot; \eta_t)$ and evaluates $f(x_i)$ for each $x_i \in X$. The sample objective values are then sorted in increasing order and are used to obtain the density parameter η_{t+1} for the next iteration by solving the following maximum likelihood estimation problem:

$$\eta_{t+1} = \arg\max_{\eta} \frac{1}{N} \sum_{i=1}^{N} I_{\{f(x_i) \geq f_{(K)}\}} \ln(d(x_i, \eta)), \tag{3}$$

where $f_{(K)}$ is the K-th largest sample objective value, with $0 < K \leq N$ being a user defined parameter. This process then repeats until the maximum number of iterations T is reached, or some convergence criterion is met.

While solving Equation (3) is generally intractable, analytic solutions exist for sampling densities of many commonly used distributions from the exponential family. For

instance, in case d is the density of a Gaussian distribution parameterized by $\eta = (\mu, \sigma^2)$, the solution of Equation (3) is $\hat{\eta} = (\hat{\mu}, \hat{\sigma}^2)$, with $\hat{\mu} = \frac{1}{|\mathcal{K}|}\sum_{x\in\mathcal{K}} x$ and $\hat{\sigma}^2 = \frac{1}{|\mathcal{K}|}\sum_{x\in\mathcal{K}}(x-\mu)^2$, where $\mathcal{K} = \{\boldsymbol{x} \in X \mid f(\boldsymbol{x}) \geq f_{(K)}\}$ are the top-K performing samples, called *elite samples*. That is, the updated distribution is a Gaussian distribution that is fitted to the elite samples. Similarly, if $\mathcal{X}$ is a multidimensional Euclidean space and d is the density of a multivariate Gaussian distribution parameterized by $\eta = (\boldsymbol{\mu}, \Sigma)$, the solution to Equation (3) is $\hat{\eta} = (\hat{\boldsymbol{\mu}}, \hat{\Sigma})$, with $\hat{\boldsymbol{\mu}} = \frac{1}{|\mathcal{K}|}\sum_{\boldsymbol{x}\in\mathcal{K}} \boldsymbol{x}$ and $\hat{\Sigma} = \frac{1}{|\mathcal{K}|}\sum_{\boldsymbol{x}\in\mathcal{K}}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^\top$.

In practice, to avoid premature convergence towards a local optimum, η is often updated according to a smoothed updating rule, i.e.,

$$\hat{\eta} = (1-\alpha)\eta + \alpha\tilde{\eta}, \tag{4}$$

where $\tilde{\eta}$ is the solution to Equation (3), and $0 < \alpha \leq 1$ is a smoothing parameter. More details on the CE-method for optimization can be found in Rubinstein and Kroese (2004); Botev et al. (2013).

Lazy Cross-Entropy Search Over Policy Trees

We introduce our method Lazy Cross-Entropy Search Over Policy Trees (LCEOPT) in this section. We first present our assumptions, followed by an overview of LCEOPT, our policy class and its parameterization, and finally how lazy policy sampling, evaluation and distribution updates are carried out. For the ease of understanding our method, we start with describing a basic method that highlights the conceptual framework of our approach but is computationally inefficient, before describing the lazy method which is much more efficient and is actually used in our LCEOPT algorithm. The key steps of LCEOPT are shown in Algorithm 1, with detailed pseudocodes provided in Appendix A [1].

Assumptions

We assume that the POMDP $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$ to be solved has a (bounded or unbounded) D-dimensional continuous action space $\mathcal{A}$, a discrete observation space $\mathcal{O}$, and an arbitrary (discrete or continuous or mixed) state space $\mathcal{S}$. Similarly to many existing online POMDP solvers, we assume access to a stochastic *generative model* $G : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{O} \times \mathbb{R}$ that simulates the transition, observation and reward models, instead of requiring an explicit representation for them. That is, for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the model G generates a next state $s' \in \mathcal{S}$, observation $o \in \mathcal{O}$ and reward $r \in \mathbb{R}$ according to $(s', o, r) = G(s, a)$, where (s', o) is distributed according to $p(s', o \mid s, a) = T(s, a, s')Z(s', a, o)$, and $r = R(s, a)$.

Overview of LCEOPT

LCEOPT is an anytime online POMDP solver based on a *lazy* CE-method that can handle incomplete data. Algorithm 1 shows the key steps of LCEOPT. Basically, at each time step (lines 2 to 8), LCEOPT estimates the optimal parameter $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} V_{\pi_{\boldsymbol{\theta}}}(b)$ of a parametric policy $\pi_{\boldsymbol{\theta}}$. Details regarding the policy parameterization are discussed in the next subsection. To this end, LCEOPT maintains and updates a distribution d on the parameter space Θ. Here, we chose the distribution to be a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$, parameterized by a mean vector $\boldsymbol{\mu}$ and a vector of variances $\boldsymbol{\sigma}^2$. The notation $\mathrm{diag}(\boldsymbol{\sigma}^2)$ denotes a diagonal covariance matrix whose main diagonal is $\boldsymbol{\sigma}^2$. While other distributions could be chosen, this particular choice enables us to derive efficient parameter sampling, evaluation and distribution update approaches. The distribution is iteratively updated using the Cross-Entropy method as described below.

Algorithm 1 LCEOPT

1: **while** problem not terminated **do**
2: Initialize current policy distribution d_0 and set $k = 0$
3: **while** planning budget not exceeded **do**
4: *Lazily* sample and evaluate candidate policies $\boldsymbol{\theta}_{1:N}$ from current policy distribution d_k
5: $\mathcal{K} \leftarrow$ Set of top-K performing parameter vectors
6: Compute the new policy distribution d_{k+1} from $\mathcal{K}$ and d_k
7: $k \leftarrow k + 1$
8: **end while**
9: $\boldsymbol{\mu} \leftarrow$ the mean of the current policy distribution
10: Execute $a^* = \pi_{\boldsymbol{\mu}}(b)$ and update belief
11: **end while**

LCEOPT starts from an initial distribution $d_0 = \mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \mathrm{diag}(\boldsymbol{\sigma}^2_{\text{init}}))$ over Θ. At each planning step k (lines 4 to 7), LCEOPT samples $N > 0$ policy parameters $\boldsymbol{\theta}_{1:N}$ from the distribution d_k and, for each sampled $\boldsymbol{\theta}_i$, computes a policy value estimate $\widehat{V}_{\boldsymbol{\theta}_i}(b) \approx V_{\pi_{\boldsymbol{\theta}_i}}(b)$ for the current belief b as the average return over multiple randomly sampled simulations. We then compute the updated distribution d_{k+1} by computing the distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ as the mean and variance of the $K > 0$ policy parameters with the highest estimated policy values. This process repeats from the updated distribution parameterized by $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ until the planning budget for the current step has been exceeded. The efficiency of the algorithm depends on how we sample and evaluate policy parameters and update the distribution parameters. In a latter section dedicated to the details for these steps, we shall describe a basic method for these steps, which serves as both a baseline and a precursor to our more efficient and novel lazy method. In the implementation of LCEOPT, we use the lazy method.

After planning ends at each time step, the action for the agent to execute is then chosen to be $a^* = \pi_{\boldsymbol{\mu}}(b)$ (lines 9 to 10). After executing the action and perceiving an observation $o \in \mathcal{O}$, we update the belief to $b' = \tau(b, a^*, o)$, where τ is the Bayesian belief update function. Our implementation uses a Sequential Importance Resampling (SIR) particle filter (Arulampalam et al. 2002) to update the belief. This process is then repeated from the updated belief until some terminal condition is satisfied (line 1).

[1] All appendices are in the full version of the paper at https://arxiv.org/abs/2305.08049

Policy Parameterization

To facilitate a simple policy parameterization and derive an efficient method to evaluate a policy, LCEOPT represents each policy π as a *policy tree* $\mathcal{T}_\pi$. From now on, we drop the subscript in $\mathcal{T}_\pi$ and implicitly assume that $\mathcal{T}$ represents policy π. A policy tree is a tree whose nodes represent actions and whose edges represent observations. It describes a decision plan, such that the agent starts by executing the action associated with the root node of $\mathcal{T}$. After perceiving an observation from the environment, the agent follows the edge representing the perceived observation, and the process repeats from the child node of the followed observation edge. For infinite-horizon problems, the depth of a policy tree may be infinite, which makes defining a suitable policy parameterization difficult. Thus, in this paper, we restrict the space of policies to be the space $\Pi_M \subset \Pi$ of all policies represented by policy trees of depth M, where $M > 0$ is a user defined parameter.

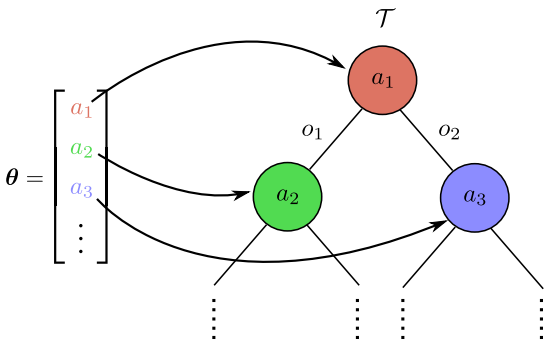

Figure 1: Illustration of the relationship between the parameter vector $\boldsymbol{\theta}$ (left) and the policy tree $\mathcal{T}$ (right), representing policy π. The components of $\boldsymbol{\theta}$ are actions that are associated with the action nodes in $\mathcal{T}$.

Policy trees provide a compact and interpretable representation of policies that give rise to a simple parameterization: A policy tree can be parameterized by a $(D\,|\mathcal{T}|)$-dimensional vector $\boldsymbol{\theta}$, such that each component $\theta_{(\nu)}$ of $\boldsymbol{\theta}$ corresponds to the action associated with a particular node $\nu \in \mathcal{T}$ in the tree. Here, D denotes the dimensionality of the action space, while $|\mathcal{T}|$ denotes the number of nodes in $\mathcal{T}$, which is equal to $(1 - |\mathcal{O}|^{M+1})/(1 - |\mathcal{O}|)$, where $|\mathcal{O}|$ is the cardinality of the observation space. Figure 1 illustrates the relationship between the parameter vector $\boldsymbol{\theta}$ and the policy tree $\mathcal{T}$.

Policy Sampling, Evaluation and Distribution Update

We first describe a basic method for sampling and evaluating policy parameters and update the distribution parameters, followed by a discussion on our lazy method.

The Basic Method Our basic method is a simple approach that highlights the conceptual framework of our LCEOPT algorithm and serves as a precursor to the more efficient lazy method described in the next subsection.

During planning, given the current policy parameter distribution $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$, we sample policy parameters $\boldsymbol{\theta}_{1:N}$ directly from the distribution. For each sampled policy $\pi_{\boldsymbol{\theta}}$, we compute an estimate of the policy value $V_{\pi_{\boldsymbol{\theta}}}(b)$ using Monte Carlo sampling. Specifically, starting from the current belief, we sample $L > 0$ state trajectories by simulating $\pi_{\boldsymbol{\theta}}$ and use the average of the accumulated discounted rewards of the trajectories as an approximation to $V_{\pi_{\boldsymbol{\theta}}}(b)$. Once a trajectory reaches a leaf node in the policy tree $\mathcal{T}_{\pi_{\boldsymbol{\theta}}}$, we use the final sampled state to compute a heuristic estimate of the optimal value $V^*(b')$, where $b' \in \mathcal{B}$ is the belief, conditioned on the action and observation sequences of the sampled trajectory. This heuristic estimate can be obtained in various ways, for instance by simulating a rollout policy, similarly to MCTS-based online POMDP solvers (Silver and Veness 2010; Seiler, Kurniawati, and Singh 2015; Sunberg and Kochenderfer 2018; Hoerger et al. 2023), or by hand-crafting estimates that exploit domain knowledge. Naturally, a good estimate of $V^*(b')$ often allows us to plan with a relatively short planning horizon while still achieving good policy performance.

After sampling and evaluating N policy parameters as above, we update the parameters of the distribution over Θ, based on the K best performing parameters (Algorithm 4 in Appendix A). In particular, we first compute new distribution parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}^2$ as the mean and variance of the elite parameter vectors. The final distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are then computed according to $\boldsymbol{\mu} \leftarrow (1-\alpha)\boldsymbol{\mu} + \alpha\tilde{\boldsymbol{\mu}}$ and $\boldsymbol{\sigma}^2 \leftarrow (1-\alpha)\boldsymbol{\sigma}^2 + \alpha\tilde{\boldsymbol{\sigma}}^2$ respectively, where $0 \leq \alpha \leq 1$ is a user-defined smoothing parameter. As discussed previously, this smooth update rule helps to avoid premature convergence of the distribution towards suboptimal regions in Θ.

The Lazy Method A key limitation of the basic method discussed in the previous section is the high computational cost of sampling full parameter vectors, particularly for high-dimensional action spaces. This is because, while sampling parameter vectors from the diagonal multivariate Gaussian distribution is simple, the large number of sampled parameter vectors can make it a computational bottleneck in the algorithm. For instance, in a problem with a 12 dimensional action space, the basic method takes tens of seconds for just one iteration of the CE-method, which makes it impractical in many applications (see our experiments).

Our lazy method provides a much more efficient way to implement the CE-method by using a key observation: large portions of a parameterized policy tree are often irrelevant when estimating its policy value, since the sampled trajectories used for the evaluation may not reach them. Based on this observation, we employ a lazy sampling method that only samples visited components of a parameter vector, where a component is visited if a sampled trajectory reaches its associated node in the policy tree.

To sample $N > 0$ new parameter vectors $\boldsymbol{\theta}_{1:N}$, we start by constructing N vectors of size $D(1-|\mathcal{O}|^{M+1})/(1-|\mathcal{O}|)$ whose elements are set to the null symbol $\varnothing$. For each policy $\pi_{\boldsymbol{\theta}}$, once a sampled trajectory reaches a node ν in the policy tree associated to $\pi_{\boldsymbol{\theta}}$, we check whether the sub-vector

$\theta_{(\nu)}$, i.e., the action associated with node ν, has already been sampled. If this is not the case, we sample a new action from the distribution $\mathcal{N}(\mu_{(\nu)}, \sigma^2_{(\nu)} I)$, which is the marginal of $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ corresponding to $\theta_{(\nu)}$, and assign the sampled action to $\theta_{(\nu)}$. Note that we sample $\theta_{(\nu)}$ only once, and keep it fixed for the remainder of the trajectory sampling process.

The above sampling method results in a set of sampled parameter vectors for which some of the components are $\varnothing$, if their corresponding nodes have never been visited. Consequently, we have to slightly modify the distribution update step of the basic method, which computes new distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ based on the elite parameter vectors $\mathcal{K}$. In particular, we update the marginal distributions corresponding to each dimension of the parameter space independently, based on the entries of the elite parameter vectors in $\mathcal{K}$ that are not $\varnothing$. That is, for the parameter dimension i, we compute the marginal distribution parameters according to

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \varnothing\}} \theta_i, \tag{5}$$

$$\tilde{\sigma}^2_i = \frac{1}{N_i} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \varnothing\}} (\theta_i - \tilde{\mu}_i)^2, \tag{6}$$

respectively[2], where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function, and $N_i = \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \varnothing\}}$ is the number of parameter vector entries along the i-th dimension that are not $\varnothing$. Note that we only update the marginal distribution parameters at the i-th dimension if $N_i > 0$. If $N_i = 0$, we simply set $\tilde{\mu}_i = \mu_i$ and $\tilde{\sigma}^2_i = \sigma^2_i$. Similarly to the basic version of the distribution update, we compute the final marginal distribution parameters according to $\mu_i \leftarrow (1-\alpha)\mu_i + \alpha\tilde{\mu}_i$ and $\sigma^2_i \leftarrow (1-\alpha)\sigma^2_i + \alpha\tilde{\sigma}^2_i$ respectively.

While the lazy algorithm is designed to speed up the basic algorithm, they usually do not compute identical results, due to different sampling and distribution update strategies. However, the lazy update algorithm can still be derived from the standard cross-entropy framework described previously, as shown by the following simple but interesting result.

Theorem 1. *Consider a set of (possibly partial) parameter vectors $\mathcal{K}$, a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$, and the the maximum likelihood estimation problem in the cross-entropy method*

$$\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^2 = \arg\max_{\boldsymbol{\mu}, \boldsymbol{\sigma}^2} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \ln(\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))), \tag{7}$$

where $N(\boldsymbol{\theta}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ denotes the marginal probability density on the non-null dimensions. Then the solution of the above problem is given by Equations (5) *and* (6)*.*

Proof. The proof follows easily from the following observation:

$$\begin{aligned}
&\sum_{\boldsymbol{\theta} \in \mathcal{K}} \ln(\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})^2)) \\
&= \sum_{\boldsymbol{\theta} \in \mathcal{K}} \sum_{i : \theta_i \neq \varnothing} \ln(\mathcal{N}(\theta_i; \mu_i, \sigma^2_i)) \\
&= \sum_{i} \sum_{\boldsymbol{\theta} \in \mathcal{K} : \theta_i \neq \varnothing} \ln(\mathcal{N}(\theta_i; \mu_i, \sigma^2_i)).
\end{aligned}$$

Thus we can estimate μ_i and σ^2_i for each dimension using the standard maximum likelihood estimates for the mean and variance of a univariate Gaussian to obtain the formulas in the theorem. □

While Theorem 1 and its proof are very simple, the result reveals an important insight that our lazy parameter sampling method enables the CE-method to be applied to incomplete data, while the standard CE-method only considers complete data.

Using the above lazy parameter sampling method can lead to significant computational savings, since only the components of the parameter vectors are sampled that are relevant for evaluating the corresponding policy tree. In our experiments, we investigate the amount of computational savings of our lazy sampling and evaluation method compared to the basic version discussed in the previous section.

Experiments and Results

We tested LCEOPT on 4 decision making problems under partial observability:

- **ContTag**: ContTag (Seiler, Kurniawati, and Singh 2015) is an extension of the popular benchmark problem Tag (Pineau, Gordon, and Thrun 2003) to continuous action spaces.
- **Pushbox2D/3D**: Pushbox2D/3D (Seiler, Kurniawati, and Singh 2015) is a scalable motion planning problem, based on air hockey in which an agent must push a disk-shape opponent into a goal area in the environment, while avoiding pushing it into a boundary region.
- **Parking2D/3D**: Parking2D/3D (Hoerger et al. 2023) is a navigation problem in which a vehicle must park between in a corridor between two obstacles, while having imperfect information regarding its starting location.
- **SensorPlacement-D**: SensorPlacement-D (Hoerger et al. 2023) is a scalable motion planning under uncertainty problem, where a manipulator with D degrees of freedom (DOF) and D revolute joints (with $D \in \{6, 8, 10, 12\}$) operates inside a 3D environment with muddy water and must attach a sensor at a marine structure while being subject to control errors.

Details regarding the problem scenarios can be found in Appendix B. The next two subsections detail the experimental setup and the results.

Experimental Setup

The purpose of our experiments is two-fold. The first one is to compare LCEOPT with three state-of-the-art

[2]Note that θ_i denotes a single number, while $\theta_{(\nu)}$ can possibly be a vector.

online POMDP solvers for continuous action spaces, POMCPOW (Sunberg and Kochenderfer 2018), VOMCPOW (Lim, Tomlin, and Sunberg 2021) and ADVT (Hoerger et al. 2023) on the above problem scenarios. To do this, we implemented LCEOPT, the tree baseline solvers POMCPOW, VOMCPOW and ADVT, and the problem scenarios in C++ using the OPPT framework (Hoerger, Kurniawati, and Elfes 2018). All evaluated solvers have parameters that need to be tuned for achieving good performance, including the depth of the lookahead trees for POMCPOW, VOMCPOW and ADVT, and the policy trees for LCEOPT respectively. To approximately determine the best parameters for each solver in the problem scenarios, we used the CE-method to search the solver's parameter spaces. The parameters for each solver and their searched value ranges are detailed in Appendix D. For each solver and problem scenario, we then used the best parameter point and ran $1,000$ simulation runs with a fixed planning time of 1s (measured in CPU time) per planning step.

The second purpose is to understand the computational benefits of our proposed lazy parameter sampling, evaluation, and distribution update method compared to the basic method. To investigate this, we implemented a variant of LCEOPT that uses the basic method and tested both variants of LCEOPT on the ContTag and SensorPlacement-12 problems. For both algorithms and problems, we measure the average CPU time required to reach 50 CE-iterations per planning step, for different sizes of the policy trees. Here, a CE-iteration refers to one iteration within the while-loop in Algorithm 1 (line 3), i.e., sampling and evaluating a set of policy parameters and updating the distribution over policy parameters. To see whether there is a notable difference in the quality of the policies computed by both algorithms, we tested them on the ContTag and SensorPlacement-6 problems, where we used a fixed number of 50 CE-iterations per planning step for both algorithms and problems. We then ran $2,000$ simulation runs for each algorithm and problem. For both algorithms, we used the same parameters that were used for comparing LCEOPT with the state-of-the-art methods.

All simulations were run single-threaded on an AMD EPYC 7003 CPU with 4GB of memory. The next section discusses the results of our experiments.

Results

Comparison with State-of-the-Art Methods Table 1 shows the average total discounted rewards of all tested solvers for the ContTag, Pushbox, Parking and SensorPlacement problems. LCEOPT outperforms the baseline solvers in all problems, except for the ContTag problem, in which ADVT performs slightly better.

Notably, LCEOPT significantly outperforms the baselines in the SensorPlacement problems. The results indicate that LCEOPT scales substantially better to higher-dimensional action spaces. For instance, in the SensorPlacement-12 problem (which consists of a 12-dimensional continuous space), LCEOPT achieves a better result than the best baseline, ADVT, in the 8-dimensional SensorPlacement-8 problem. A similar effect can be seen in the Parking problems, where the performance of LCEOPT suffers only marginally compared to the baselines as the dimensionality of the action space increases.

We conjecture that this is due to the action sampling strategies of the baselines. POMCPOW uses a simple uniform action sampling strategy, which does not take the value of already sampled actions into account. ADVT and VOMCPOW construct Voronoi cells in the action space at each sampled belief and bias their action sampling strategies towards cells with good performing representative actions. However, for higher-dimensional action spaces, these cells may be too large to quickly focus sampling towards near optimal regions in the action space. On the other hand, LCEOPT is a partition-free method which uses distributions over the policy space. At each node in the policy trees, LCEOPT maintains and updates a sampling distribution that quickly focuses its probability mass towards near-optimal regions in the action space. This property allows LCEOPT to scale much more effectively to higher-dimensional action spaces, compared to the baselines.

Another interesting observation is that all solvers require only a relatively short planning horizon for most of the problem scenarios. For the ContTag, Pushbox and SensorPlacement problems, all solvers require an effective planning horizon of only two steps. The reason is that all solvers use heuristic estimates of $V^*(b)$ when the planning horizon is reached, that is, a leaf node in the policy trees of LCEOPT and lookahead trees of POMCPOW, VOMCPOW and ADVT is reached. For all problem scenarios, we designed simple state-dependent heuristic estimates of $V^*(b)$ by removing partial observability and action noise from the problem. Details regarding the heuristic estimates are provided in Appendix C. Such simple heuristics are often useful in keeping the required effective planning horizon short. For the Parking problems, we require a slightly longer effective planning horizon of five steps to achieve good performance, because actions have potentially long-term consequences. For instance, if the vehicle decides to accelerate aggressively while navigating towards the goal, it may require multiple steps to decelerate in order to avoid crashing into an obstacle. Such long-term consequences of actions are often difficult to capture via simple state-dependent heuristics, leading to a longer effective planning horizon required to find good solutions.

Comparison of the basic Method and the Lazy Method Table 2 shows the average CPU time (measured in seconds) required for LCEOPT with the lazy and the basic parameter sampling method to reach 50 CE-iterations per planning step for the ContTag and SensorPlacement-12 problems respectively, as we increase the policy tree depth M. It can be seen that for the ContTag problem, both the lazy and basic methods perform similar for more shallower trees (up to $M = 4$), while the lazy method performs slightly better for policy trees of depth $M = 5$. However, the lazy method outperforms the basic one significantly in the SensorPlacement-12 problem, even for shallow policy trees. The reason is that the dimensionality of the parameter space increases dramatically for deeper policy trees, due to the 12-dimensional ac-

	ContTag	Pushbox2D	Pushbox3D	Parking2D	Parking3D
LCEOPT (Ours)	0.02 ± 0.23	$\mathbf{399.7 \pm 8.7}$	$\mathbf{358.6 \pm 12.3}$	$\mathbf{53.4 \pm 0.4}$	$\mathbf{47.2 \pm 0.6}$
ADVT	$\mathbf{0.37 \pm 0.18}$	356.9 ± 9.9	327.8 ± 14.7	43.1 ± 2.1	34.6 ± 2.1
VOMCPOW	-1.95 ± 0.31	323.5 ± 12.8	145.7 ± 13.7	1.3 ± 1.9	-11.7 ± 1.3
POMCPOW	-2.00 ± 0.31	96.7 ± 15.4	25.9 ± 12.2	-3.9 ± 1.8	-18.4 ± 1.1

	SensorPlacement-6	SensorPlacement-8	SensorPlacement-10	SensorPlacement-12
LCEOPT (Ours)	$\mathbf{914.3 \pm 2.6}$	$\mathbf{885.5 \pm 2.9}$	$\mathbf{858.8 \pm 4.2}$	$\mathbf{832.1 \pm 4.5}$
ADVT	859.2 ± 12.2	794.1 ± 15.3	631.4 ± 23.9	456.8 ± 28.2
VOMCPOW	754.4 ± 12.8	540.5 ± 17.2	276.8 ± 17.8	73.6 ± 12.1
POMCPOW	354.5 ± 19.9	124.2 ± 15.3	12.2 ± 8.2	-6.0 ± 4.9

Table 1: Average total discounted rewards and 95% confidence intervals of all tested solvers for the ContTag, Pushbox, Parking and SensorPlacement problems. The average is taken over $1,000$ simulation runs per solver and problem, with a planning time of 1s per step. The best result for each problem scenario is highlighted in bold.

		$M=1$	$M=2$	$M=3$	$M=4$	$M=5$
ContTag	Lazy	0.43	0.64	0.90	1.19	1.39
	Basic	0.43	0.64	0.91	1.23	1.57
SensorPlacement-12	Lazy	1.09	1.58	2.2	3.07	5.13
	Basic	2.65	12.61	137.56	900.97	3928.45

Table 2: Comparison of the time efficiency of the basic and the lazy policy sampling strategies on the ContTag and SensorPlacement-12 problems. The table shows the average CPU time (in seconds) to reach 50 CE-iterations for different policy tree depths. The average is taken over 20 planning steps. Larger values indicate a larger parameter sampling cost. For the ContTag problem, we set the number of candidate policies to $N = 493$ and the number of trajectories per parameter vector to $L = 103$ for both algorithms. For the SensorPlacement-12 problem, we set $N = 496$ and $L = 11$.

	ContTag	SensorPlacement-6
Lazy	0.15 ± 0.16	920.4 ± 1.8
Basic	-0.11 ± 0.16	919.8 ± 1.8

Table 3: Average total discounted rewards and 95% confidence intervals of LCEOPT using the lazy and the basic policy sampling strategies in the ContTag and SensorPlacement-12 problems. For both algorithms, we use 50 CE-iterations per planning step. The average is taken over 2,000 simulation runs for both algorithms and problems.

tion space. As a consequence, sampling full parameter vectors becomes computationally too expensive. On the other hand, our lazy method only samples the components of the parameter vectors that are relevant to evaluate the associated policy. The number of relevant components of a parameter vector is typically much smaller than the dimensionality of the parameter space, which leads to significant computational savings when sampling parameter vectors lazily.

Table 3 shows the average total discounted rewards for both the lazy and the basic version of LCEOPT in the ContTag and SensorPlacement-6 problems, where for both variants, the number of CE-iterations per planning step is set to 50. It can be seen that despite the different policy distribution update behaviors, both algorithms perform similar in the ContTag and SensorPlacement problems. This indicates that the lazy algorithm is able to retain the good performance of the basic one, while being much more efficient computationally.

Conclusion

Online POMDP solvers have seen tremendous progress in the last two decades in solving increasingly complex decision making under uncertainty problems. Despite this progress, solving continuous-action POMDPs remains a challenge. In this paper, we propose a simple online POMDP solver, called Lazy Cross-Entropy Search Over Policy Trees (LCEOPT) designed for POMDP problems with continuous state and action spaces. LCEOPT uses a lazy version of the CE-method on the space of policy trees to find a near-optimal policy. Despite its simple structure, LCEOPT shows a strong empirical performance against state-of-the-art methods on four benchmark problems, particularly on those with higher-dimensional action spaces. These results indicate that gradient-free optimization methods that do not rely on partitioning the search space are viable tools for solving continuous POMDPs.

An interesting avenue for future work is to generalize our method to POMDPs with continuous observation spaces. This would allow us to consider an even larger class of POMDPs. In addition, our lazy CE method can be used to handle problems where the objective function can be evaluated using just a subset of the parameters. We can perform lazy sampling to sample only the relevant parameters, and perform distribution update by maximizing the marginal likelihood of the partial parameter vectors. This may be useful in other applications.

Acknowledgements

This work is partially supported by the Australian Research Council (ARC) Discovery Project 200101049.

References

Agha-mohammadi, A.-a.; Chakravorty, S.; and Amato, N. M. 2011. FIRM: Feedback controller-based information-state roadmap - A framework for motion planning under uncertainty. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4284–4291.

Arulampalam, M.; Maskell, S.; Gordon, N.; and Clapp, T. 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2): 174–188.

Bai, H.; Hsu, D.; and Lee, W. S. 2014. Integrated perception and planning in the continuous space: A POMDP approach. *The International Journal of Robotics Research*, 33(9): 1288–1302.

Botev, Z. I.; Kroese, D. P.; Rubinstein, R. Y.; and L'Ecuyer, P. 2013. The Cross-Entropy Method for Optimization. In Rao, C.; and Govindaraju, V., eds., *Handbook of Statistics - Machine Learning: Theory and Applications*, 35–59. Elsevier.

Coulom, R. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In van den Herik, H. J.; Ciancarini, P.; and Donkers, H. H. L. M. J., eds., *Computers and Games*, 72–83. Springer. ISBN 978-3-540-75538-8.

de Boer, P.-T.; Kroese, D. P.; Mannor, S.; and Rubinstein, R. Y. 2005. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1): 19–67.

Filar, J. A.; Qiao, Z.; and Ye, N. 2019. POMDPs for sustainable fishery management. In *23rd International Congress on Modelling and Simulation-Supporting Evidence-Based Decision Making: The Role of Modelling and Simulation, MODSIM 2019*, 645–651. Modelling and Simulation Society of Australia and New Zealand.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning Latent Dynamics for Planning from Pixels. In *Proc. of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2555–2565. PMLR.

Hoerger, M.; Kurniawati, H.; Bandyopadhyay, T.; and Elfes, A. 2020. Linearization in Motion Planning under Uncertainty. In Goldberg, K.; Abbeel, P.; Bekris, K.; and Miller, L., eds., *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, 272–287. Springer. ISBN 978-3-030-43089-4.

Hoerger, M.; Kurniawati, H.; and Elfes, A. 2018. A Software Framework for Planning Under Partial Observability. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–9.

Hoerger, M.; Kurniawati, H.; Kroese, D.; and Ye, N. 2023. Adaptive Discretization Using Voronoi Trees for Continuous-Action POMDPs. In LaValle, S. M.; O'Kane, J. M.; Otte, M.; Sadigh, D.; and Tokekar, P., eds., *Algorithmic Foundations of Robotics XV: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, 170–187. Springer. ISBN 978-3-031-21090-7.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2): 99–134.

Kurniawati, H. 2022. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5: 253–277.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. S. 2010. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3): 308–323.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems III*, 65–72. MIT Press.

Kurniawati, H.; and Yadav, V. 2016. An online POMDP solver for uncertainty planning in dynamic environment. In *Robotics Research: The 16th International Symposium ISRR*, 611–629. Springer.

Lim, M. H.; Tomlin, C. J.; and Sunberg, Z. N. 2021. Voronoi Progressive Widening: Efficient Online Solvers for Continuous State, Action, and Observation POMDPs. In *2021 60th IEEE Conference on Decision and Control (CDC)*, 4493–4500.

Lindquist, A. 1973. On Feedback Control of Linear Stochastic Systems. *SIAM Journal on Control*, 11(2): 323–343.

Mannor, S.; Rubinstein, R. Y.; and Gat, Y. 2003. The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning*, 512–519. AAAI Press.

Mern, J.; Yildiz, A.; Sunberg, Z.; Mukerji, T.; and Kochenderfer, M. J. 2021. Bayesian optimized Monte Carlo planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11880–11887. AAAI.

Oliehoek, F. A.; Kooij, J. F. P.; and Vlassis, N. 2008. A Cross-Entropy Approach to Solving Dec-POMDPs. In Badica, C.; and Paprzycki, M., eds., *Advances in Intelligent and Distributed Computing*, 145–154. Springer. ISBN 978-3-540-74930-1.

Omidshafiei, S.; Agha-mohammadi, A.-a.; Amato, C.; Liu, S.-Y.; How, J. P.; and Vian, J. 2016. Graph-based Cross Entropy method for solving multi-robot decentralized POMDPs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5395–5402.

Papadimitriou, C. H.; and Tsitsiklis, J. N. 1987. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3): 441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, 1025–1032.

Rubinstein, R. Y.; and Kroese, D. P. 2004. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte Carlo Simulation and Machine Learning*. Springer. ISBN 038721240X.

Schwartz, J.; Kurniawati, H.; and El-Mahassni, E. 2020. POMDP+ information-decay: Incorporating defender's behaviour in autonomous penetration testing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 235–243.

Seiler, K. M.; Kurniawati, H.; and Singh, S. P. 2015. An online and approximate solver for POMDPs with continuous action space. In *2015 IEEE International Conference on Robotics and Automation*, 2290–2297. IEEE.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., *Advances in Neural Information Processing Systems*, volume 23. Curran Associates.

Smith, T.; and Simmons, R. 2005. Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 542–549. AUAI Press. ISBN 0974903914.

Sun, W.; Patil, S.; and Alterovitz, R. 2015. High-Frequency Replanning Under Uncertainty Using Parallel Sampling-Based Motion Planning. *IEEE Transactions on Robotics*, 31(1): 104–116.

Sunberg, Z.; and Kochenderfer, M. 2018. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 259–263. AAAI Press.

van den Berg, J.; Abbeel, P.; and Goldberg, K. 2011. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7): 895–913.

van den Berg, J.; Patil, S.; and Alterovitz, R. 2012. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11): 1263–1278.

Wang, E.; Kurniawati, H.; and Kroese, D. 2018. An On-Line Planner for POMDPs with Large Discrete Action Space: A Quantile-Based Approach. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 273–277. AAAI Press.

Ye, N.; Somani, A.; Hsu, D.; and Lee, W. S. 2017. DESPOT: Online POMDP Planning with Regularization. *Journal of Artificial Intelligence Research*, 58: 231–266.