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

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

[1]School of Mathematics and Physics, The University of Queensland, Brisbane, Queensland, Australia.
[2]School of Computing, Australian National University, Canberra, ACT, Australia.

*Corresponding author(s). E-mail(s): m.hoerger@uq.edu.au;
Contributing authors: 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 *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.

Keywords: continuous-action POMDP, cross-entropy method, policy tree, lazy computation

1 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, & Ye, 2019) and cyber security (Schwartz, Kurniawati, & El-Mahassni, 2020). While an autonomous agent interacts with the environment, the exact state of the agent and/or the state of the environment is often only partially observed, due to stochastic effects of actions, noisy observations and incomplete information regarding the environment. Thus, in order to make optimal decisions, the agent must take the these uncertainties into account during planning. The Partially Observable Markov Decision Process (POMDP) is a principled framework to solve such decision making problems in stochastic partially observable environments. POMDPs lift the problem from the agent's state space to its *belief space*, i.e., the space of all probability distributions over the state space. This enables autonomous agents to systematically account for uncertain action-effects and incomplete or noisy observations while computing an optimal strategy, called a *policy*. Although it has been shown that solving POMDPs exactly is computationally intractable in general (Papadimitriou & Tsitsiklis, 1987), many efficient sampling-based POMDP solvers have been developed in the past two decades (reviewed in Kurniawati (2022)), whose idea is to trade optimality with computational tractability. This has enabled POMDPs to become viable tools for many realistic decision making problems under uncertainty.

Despite these advances, solving POMDPs with continuous action spaces remains challenging, particularly for high-dimensional action spaces. Current state-of-the-art solvers for POMDPs with continuous action spaces (Hoerger, Kurniawati, Kroese, & Ye, 2023; Lim, Tomlin, & Sunberg, 2021; Mern, Yildiz, Sunberg, Mukerji, & Kochenderfer, 2021; Seiler, Kurniawati, & Singh, 2015; Sunberg & Kochenderfer, 2018) typically aim to solve the problem online; interleaving the computation of a near-optimal strategy and the execution of the strategy. These online solvers commonly 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 a partitioning of the action space. In contrast, our method uses a stochastic optimization approach in the policy space by utilizing the Cross-Entropy method (de Boer, Kroese, Mannor, & Rubinstein, 2005; Rubinstein & Kroese, 2004), while avoiding any form of partitioning of the action space.

Apart from the above MCTS-based methods, some approaches have been proposed that utilize the Cross-Entropy method for solving POMDPs or MDPs (the fully-observable variant of POMDPs). For MDPs, Mannor, Rubinstein,

and Gat (2003) proposed a method that uses the Cross-Entropy method to optimize the policy for discrete-action MDPs. For DEC-POMDPs, a variant of POMDPs in multi-agent settings, Oliehoek, Kooij, and Vlassis (2008) proposed a Cross-Entropy based solver that optimizes over the joint-policies of multiple agents in the environment. However, their method assumes discrete state, action and observation spaces. More recently, Wang, Kurniawati, and Kroese (2018) proposed QBASE, a method that combines Cross-Entropy based optimisation with MCTS to solve POMDPs with large discrete action spaces. Unlike the above methods, which have all been designed for problems with discrete action spaces, our method considers POMDPs with continuous action spaces. Omidshafiei et al. (2016) proposed an online solver for DEC-POMDPs, based on Cross-Entropy optimization over macro-actions, represented as Finite State Automata (FSA). While this method considers continuous action spaces, the optimization is carried out over a finite policy space. In another line of work, Hafner et al. (2019) proposed a Cross-Entropy based POMDP solver within a deep planning framework that optimizes over open-loop action sequences, while our method optimizes over the policy space.

To approximately solve POMDPs with continuous state and action spaces, we propose a new online POMDP solvers, called Lazy Cross-Entropy Search Over Policy Trees (LCEOPT). The key idea of LCEOPT is to frame the problem of computing a near-optimal policy as a stochastic optimization problem in the policy space and to extend the Cross-Entropy method for optimization to approximately solve this problem. To do this, 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 the MCTS-based online solvers discussed above, 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).

2 Background & Related Work

We provide a brief introduction to POMDP in Section 2.1, followed by a discussion of current POMDP solvers in Section 2.2. Section 2.3 gives a brief overview of the Cross-Entropy method for optimization.

2.1 Partially Observable Markov Decision Process (POMDP)

A POMDP provides a general mathematical framework for sequential decision making under uncertainty, in which an autonomous agent operates in an environment, while having imperfect knowledge regarding its state or the state of the environment.

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 $b_0 \in \mathcal{B}$, which is 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, $T(s_t, a_t, s_{t+1})$ represents a probability mass function, whereas for continuous state spaces, it represents 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 $Z(s_{t+1}, a_t, o_t) = p(o_t|s_{t+1}, a_t)$. Here, $Z(s_{t+1}, a_t, o_t)$ represents a probability mass function for discrete observation spaces, or a probability density function for continuous observation spaces respectively. 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

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

In designing solvers, it is often convenient to work with the *action value* or *Q-value*

$$Q(b, a) = R(b, a) + \gamma \mathbb{E}_{o \in \mathcal{O}}[V_{\pi^*}(b_a^o) \,|\, b], \tag{3}$$

where $R(b,a) = \int_{s \in \mathcal{S}} b(s)R(s,a)\mathrm{d}s$ is the expected reward of executing action a at belief b and $b_a^o = \tau(b,a,o)$ is the updated agent's belief estimate after it performs action $a \in \mathcal{A}$ while at belief b, and subsequently perceives observation $o \in \mathcal{O}$. The optimal value function is then

$$V^*(b) = \max_{a \in \mathcal{A}} Q(b,a). \tag{4}$$

A more elaborate explanation is available in Kaelbling, Littman, and Cassandra (1998).

2.2 POMDP Solvers

Various efficient sampling-based offline and online POMDP solvers have been developed for increasingly complex discrete and continuous POMDPs in the last two decades. Offline solvers (e.g., (Bai, Hsu, & Lee, 2014; Kurniawati, Du, Hsu, & Lee, 2010; Kurniawati, Hsu, & Lee, 2008; Pineau, Gordon, & Thrun, 2003; Smith & Simmons, 2005)) compute an approximately optimal policy for all beliefs first, before deploying it for execution. In contrast, online solvers (e.g., (Kurniawati & Yadav, 2016; Silver & Veness, 2010; Ye, Somani, Hsu, & Lee, 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. For instance, POMCPOW (Sunberg & Kochenderfer, 2018) uses Progressive Widening (Couëtoux, Hoock, Sokolovska, Teytaud, & Bonnard, 2011) in conjunction with a uniform action sampling strategy to continuously add new actions to the search tree that is built using MCTS. VOMCPOW (Lim et al., 2021) replaces POMCPOW's uniform action sampling strategy with a more sophisticated strategy (Kim, Lee, Lim, Kaelbling, & Lozano-Pérez, 2020) that constructs a Voronoi diagram in the action space and biases action sampling towards Voronoi cells with high-performing actions. ADVT (Hoerger et al., 2023) treats the problem of computing a near-optimal action as a continuum-arm bandit problem and constructs a hierarchical partitioning of the action space to bias action sampling and action selection during planning. While VOMCPOW and ADVT have been shown to be effective for problems with lower-dimensional continuous action spaces (up to 10-dimensional action spaces), their performance tends to degrade for problems with higher-dimensional action spaces due to the need to partition the action space.

In addition to the MCTS-based solvers discussed above and the Cross-Entropy-based methods discussed in Section 1, some solvers (Aghamohammadi, Chakravorty, & Amato, 2011; Sun, Patil, & Alterovitz, 2015; van den Berg, Abbeel, & Goldberg, 2011; van den Berg, Patil, & 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, Kurniawati, Bandyopadhyay, & Elfes, 2020). In contrast, our method requires no restriction on the class of beliefs, while simultaneously retaining efficiency in higher-dimensional action spaces.

2.3 Cross-Entropy Method for Optimization

The Cross-Entropy (CE) Method (Botev, Kroese, Rubinstein, & L'Ecuyer, 2013; Rubinstein & Kroese, 2004) is a gradient-free method for discrete and continuous optimization problems with either deterministic or noisy objective functions. Suppose $\mathcal{X}$ is an arbitrary set of states, 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 set of samples $X = \{x_i\}_{i=1}^N$ 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:

$$\eta_{t+1} = \arg\max_{\eta} \frac{1}{N} \sum_{i=1}^{N} I_{\{f(x_i) \geq f_{(K)}\}} \ln(d(x_i, \eta)), \tag{5}$$

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 iterations T is reached, or some convergence criterion is met.

While solving eq. (5) is generally intractable, analytic solutions exist when the sampling density d is chosen from an exponential family (in case $\mathcal{X}$ is continuous). For instance, in case d is the density of a Gaussian distribution parameterized by $\eta = (\mu, \sigma^2)$, the solution of eq. (5) 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 space and d is the density of a multivariate Gaussian distribution parameterized by $\eta = (\boldsymbol{\mu}, \Sigma)$, the solution to eq. (5) 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.,

$$\hat{\eta} = (1 - \alpha)\eta + \alpha\tilde{\eta}, \tag{6}$$

where $\tilde{\eta}$ is the solution to eq. (5), and $0 < \alpha \leq 1$ is a smoothing parameter.

More details on the CE-method for optimization can be found in Botev et al. (2013); Rubinstein and Kroese (2004).

3 Lazy Cross-Entropy Search Over Policy Trees

We present the assumptions and an overview of our method Lazy Cross-Entropy Search Over Policy Trees (LCEOPT) in Section 3.1 and Section 3.2 respectively, and then present the details in the following subsections: we first describe our policy class and its parameterization in Section 3.3, then describe how policy sampling, evaluation and distribution update are carried out in Section 3.4. Specifically, Section 3.4.1 describes a basic method that highlights the conceptual framework of our approach but is computationally inefficient. Section 3.4.2 describes a lazy method that is much more efficient and is actually used in our LCEOPT algorithm.

3.1 Assumptions

We assume that in the POMDP $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$ to be solved, the action space $\mathcal{A}$ is a D-dimensional continuous space that can be either bounded or unbounded, the observation space $\mathcal{O}$ is discrete, and the state space $\mathcal{S}$ can be either discrete or continuous or mixed. Similar to many existing online POMDP solvers, instead of requiring an explicit representation of the transition, observation and reward functions T, Z and R, we assume that we have access to a *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. That is, for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the model G simulates 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 $p(s', o \,|\, s, a) = T(s, a, s')Z(s', a, o)$, and $r = R(s, a)$.

3.2 Overview of LCEOPT

LCEOPT is an anytime online POMDP solver. The main idea of LCEOPT is to extend the CE-method to solving continuous action POMDPs. To do this, LCEOPT utilizes a policy parameterization, such that each policy $\pi \in \Pi$ is uniquely defined by a parameter vector $\boldsymbol{\theta} \in \Theta$, where Θ is the parameter space. Thus, the problem of computing an approximation to the optimal policy π^* amounts to estimating the parameter $\boldsymbol{\theta}^* \in \Theta$, such that $V_{\pi_{\boldsymbol{\theta}^*}}(b) = V_{\pi^*}(b)$. LCEOPT applies the CE-method to estimate $\boldsymbol{\theta}^*$ by maintaining a distribution over Θ. Here we chose the distribution to be a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$, parameterized by a mean vector $\boldsymbol{\mu}$ and a vector of variances $\boldsymbol{\sigma}^2$. The notation $\text{diag}(\boldsymbol{\sigma}^2)$ is 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, as we will discuss in Section 3.4.2. The distribution is iteratively updated by sampling a set of parameters from the

distribution and estimating $V_{\pi_{\boldsymbol{\theta}}}(b)$ for each sampled parameter $\boldsymbol{\theta}$, where $\pi_{\boldsymbol{\theta}}$ is the policy parameterized by $\boldsymbol{\theta}$. Details regarding the policy parameterization are discussed in Section 3.3.

An overview of LCEOPT is shown in Algorithm 1. LCEOPT starts from an

Algorithm 1 LCEOPT(Initial belief b_0, number of candidate policies per iteration $N > 0$, number of elite samples $K > 0$, number of trajectories $L > 0$, initial distribution parameters $(\boldsymbol{\mu}_{\text{init}}, \boldsymbol{\sigma}^2_{\text{init}})$, smoothing parameter $0 < \alpha \leq 1$)

1: $b \leftarrow b_0$
2: isTerminal $\leftarrow$ False
3: **while** isTerminal is False **do**
4: $\quad \boldsymbol{\mu} \leftarrow \boldsymbol{\mu}_{\text{init}}, \quad \boldsymbol{\sigma}^2 \leftarrow \boldsymbol{\sigma}^2_{\text{init}}$
5: $\quad$ **while** planning budget not exceeded **do**
6: $\quad\quad$ **for** $i = 1$ to N **do**
7: $\quad\quad\quad$ // Sample and evaluate a candidate policy
8: $\quad\quad\quad (\boldsymbol{\theta}_i, \widehat{V}_i(b)) \leftarrow$ SampleAndEvaluatePolicy$(b, (\boldsymbol{\mu}, \boldsymbol{\sigma}^2), L)$ ▷ Algorithm 4
9: $\quad\quad$ **end for**
10: $\quad\quad$ // Sort evaluated parameters in increasing order according to their estimates values $\widehat{V}$
11: $\quad\quad \mathcal{K} \leftarrow$ Set of top-K performing parameter vectors
12: $\quad\quad$ // Update the distribution parameters
13: $\quad\quad (\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \leftarrow$ UpdateDistribution$((\boldsymbol{\mu}, \boldsymbol{\sigma}^2), \mathcal{K}, \alpha)$ ▷ Algorithm 5
14: $\quad$ **end while**
15: $\quad a^* \leftarrow \pi_{\boldsymbol{\mu}}(b)$
16: $\quad (o,$ isTerminal$) \leftarrow$ Execute a^*
17: $\quad b' \leftarrow \tau(b, a^*, o)$
18: $\quad b \leftarrow b'$
19: **end while**

initial distribution $\mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{init}}))$ over Θ (line 4). At each planning step, LCEOPT samples $N > 0$ policy parameters $\{\boldsymbol{\theta}_i\}_{i=1}^N$ from the distribution and, for each sampled $\boldsymbol{\theta}_i$, computes $\widehat{V}_{\boldsymbol{\theta}_i}(b) \approx V_{\pi_{\boldsymbol{\theta}_i}}(b)$, i.e., an approximation to the value of the policy $\pi_{\boldsymbol{\theta}_i}$, starting from the current belief (lines 6 to 9). Given the sampled policy parameters and their corresponding estimated policy values, we update the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ parameters of the distribution. In particular, we sort the sampled policy parameters in increasing order according their corresponding estimated policy values, and keep the $K > 0$ best performing (i.e., elite) policy parameters (line 11). We then update the distribution over Θ (line 13) by computing new $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ parameters based on the mean and variance of the elite samples. This process then repeats from the distribution parameterized by the updated $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ until the planning budget for the current step has been exceeded. Section 3.4.1 describes a basic method to sample and evaluate policy parameters and update the distribution parameters, which serves as

both a baseline and a precursor to our more efficient method presented in Section 3.4.2. In the implementation of LCEOPT, we use the methods in Section 3.4.2.

The action for the agent to execute is then chosen to be $a^* = \pi_{\boldsymbol{\mu}}(b)$ (line 15). After executing the action and perceiving an observation $o \in \mathcal{O}$, we update the belief to $b' = \tau(b, a^*, o)$ (line 17), where τ is the Bayesian belief update function. In practice, we use a Sequential Importance Resampling (SIR) particle filter (Arulampalam, Maskell, Gordon, & Clapp, 2002) to update the belief. This process is then repeated from the updated belief until some terminal condition is satisfied.

3.3 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. Note that in theory, the depth of a policy tree is 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.

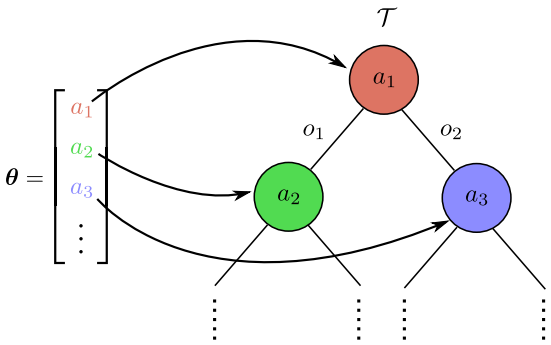


Fig. 1 Illustration of the relationship between the parameter vector $\boldsymbol{\theta}$ (left) and the policy tree $\mathcal{T}$ (right), representing policy π. The components of $\boldsymbol{\theta}$ are actions that are associated with the action nodes in $\mathcal{T}$.

Policy trees provide a compact and interpretable representation of policies that give rise to a simple parameterization: A policy tree can be uniquely parameterized by a $(D\,|\mathcal{T}|)$-dimensional vector $\boldsymbol{\theta}$, such that each component

$\theta_{(\nu)}$[1] 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 $\mathcal{T}$.

3.4 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.

3.4.1 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 describe in the next subsection.

Algorithm 2 presents the basic method. To sample a policy, we sample a parameter vector $\boldsymbol{\theta}$ according to $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ (line 2), given the current distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. As discussed in the previous section, $\boldsymbol{\theta}$ uniquely parameterizes a policy tree $\mathcal{T}$. Given the policy tree $\mathcal{T}$, we approximate the value $V_\pi(b)$ of its associated policy π for the current belief b using Monte Carlo sampling. In particular, we sample $L > 0$ reward trajectories, starting from the current belief by simulating the policy encoded in $\mathcal{T}$ and use the average of the accumulated discounted rewards of the trajectories as an approximation to $V_\pi(b)$.

To sample a reward trajectory, we first set the current node $\nu \in \mathcal{T}$ to be the root of the policy tree (line 5), and sample a state $s \in \mathcal{S}$ from the current belief (line 9). Given the sampled state, we simulate the action associated to ν, i.e., the parameter vector component $\theta_{(\nu)}$ (line 11), via the generative model G to sample a next state $s' \in \mathcal{S}$, observation $o \in \mathcal{O}$ and immediate reward r (line 13). We then set ν to be the node whose parent edge is the sampled observation (line 14). This process repeats until we reach a terminal state (line 16), or an action associated to a leaf node in $\mathcal{T}$ has been simulated. In the latter case, we compute a problem-dependent heuristic estimate of the value $V^*(b')$ (line 22), given the final sampled state, where $b' \in \mathcal{B}$ is the belief, conditioned on the action and observation sequences of the sampled trajectory. This heuristic estimate serves a similar role as the rollout policy in MCTS-based online POMDP solvers (Hoerger et al., 2023; Seiler et al., 2015; Silver & Veness, 2010; Sunberg & Kochenderfer, 2018) and provides LCEOPT with a value estimate of the optimal policy beyond the reached leaf node of $\mathcal{T}$. Interestingly, as we will see in Section 4, a good estimate of $V^*(b')$ often allows us to plan with a relatively short planning horizon while still achieving good policy performance.

[1] $\theta_{(\nu)}$ denotes the component vector of $\boldsymbol{\theta}$ associated to node ν, while θ_i denotes the parameter value of $\boldsymbol{\theta}$ at the i-th dimension.

Algorithm 2 SampleAndEvaluatePolicyBasic(Belief b, distribution parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, number of trajectories L)

1: // Sample parameter vector $\boldsymbol{\theta}$ from a Multivariate Normal distribution parameterized by $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$.
2: $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$
3: $\mathcal{T} \leftarrow$ Construct policy tree parameterized by $\boldsymbol{\theta}$
4: $M \leftarrow$ Depth of $\mathcal{T}$
5: $\nu \leftarrow$ Root node of $\mathcal{T}_{\boldsymbol{\theta}}$
6: **for** $l = 1$ to L **do**
7: isTerminal $\leftarrow$ False
8: // Sample an initial state from b.
9: $s \sim b$
10: **for** $m = 1$ to M **do**
11: $a \leftarrow \theta_{(\nu)}$
12: // Sample a next state s', observation o and immediate reward r_m from the generative model G.
13: $(s', o, r_m) \leftarrow G(s, a)$
14: $\nu \leftarrow$ Child node of ν via observation edge o
15: $s \leftarrow s'$
16: **if** s is terminal **then**
17: isTerminal $\leftarrow$ True
18: **break**
19: **end if**
20: **end for**
21: **if** isTerminal = False **then**
22: $r_{M+1} \leftarrow$ Heuristic(s)
23: **else**
24: $r_{M+1} \leftarrow 0$
25: **end if**
26: // Accumulated total discounted reward of trajectory l.
27: $R_l \leftarrow \sum_{m=1}^{M+1} \gamma^{m-1} r_m$
28: **end for**
29: $V \leftarrow \frac{1}{L} \sum_{l=1}^{L} R_l$
30: **return** $(\boldsymbol{\theta}, V)$

Finally, we compute the accumulated total discounted reward of the trajectory (line 27). The average of the accumulated total discounted rewards of all sampled trajectories then provides us with an approximation to $V_\pi(b)$ (line 29).

After sampling and evaluating N policy parameters using the method above, we update the parameters of the distribution over Θ, based on the K best performing parameters as shown in Algorithm 3. In particular, we compute new distribution parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}^2$ as the mean an variance of the elite parameter vectors (line 1). 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 $\boldsymbol{\sigma}^2 \leftarrow (1-\alpha)\boldsymbol{\sigma}^2 + \alpha\tilde{\boldsymbol{\sigma}}^2$

Algorithm 3 UpdateDistributionBasic(Distribution parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, elite samples $\mathcal{K}$, smoothing parameter α)

1: $\tilde{\boldsymbol{\mu}} \leftarrow \frac{1}{|\mathcal{K}|} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \boldsymbol{\theta}$; $\quad \tilde{\boldsymbol{\sigma}}^2 \leftarrow \frac{1}{|\mathcal{K}|} \sum_{\boldsymbol{\theta} \in \mathcal{K}} (\boldsymbol{\theta} - \tilde{\boldsymbol{\mu}})^2$
2: $\boldsymbol{\mu} \leftarrow (1 - \alpha)\boldsymbol{\mu} + \alpha \tilde{\boldsymbol{\mu}}$
3: $\boldsymbol{\sigma}^2 \leftarrow (1 - \alpha)\boldsymbol{\sigma}^2 + \alpha \tilde{\boldsymbol{\sigma}}^2$
4: **return** $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

respectively (lines 2 and 3), where $0 \leq \alpha \leq 1$ is a user-defined smoothing parameter. As discussed in Section 2.3, this smooth update rule helps to avoid premature convergence of the distribution towards sub-optimal regions in Θ.

3.4.2 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 Section 4.3.2).

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 never reach them. Based on this observation, we employ a lazy sampling method that only samples visited components of a parameter vector. Here, a component is visited if a sampled trajectory reaches its associated node in the policy tree.

Our lazy approach is shown in Algorithm 4. To sample a new parameter vector $\boldsymbol{\theta}$, we start by constructing a vector of size $D(1 - |\mathcal{O}|^{M+1})/(1 - |\mathcal{O}|)$ whose elements are set to $\emptyset$ (line 1). When a sampled trajectory reaches a node $\nu \in \mathcal{T}$ in the policy tree, we check whether the parameter vector component $\theta_{(\nu)}$, i.e., the action associated with node ν has already been sampled (line 10). 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 $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ corresponding to $\theta_{(\nu)}$, and assign the sampled action to $\theta_{(\nu)}$ (lines 12 and 13). Note that we sample $\theta_{(\nu)}$ only once, and keep it fixed for the remainder of the trajectory sampling process.

The result of the above sampling method is a set of sampled parameter vectors for which some of the components are $\emptyset$, if their corresponding nodes have never been visited. As a consequence, we have to slightly modify the distribution update step in Algorithm 3 which computes new distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ based on the elite parameter vectors $\mathcal{K}$. The modified update step is shown in Algorithm 5. In particular, we update the marginal distributions corresponding to each dimension of the parameter space independently, based

Algorithm 4 SampleAndEvaluatePolicy(Belief b, Distribution parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma})$, Number of trajectories L)

1: $\boldsymbol{\theta} \leftarrow$ Construct empty parameter vector
2: $\mathcal{T}_{\boldsymbol{\theta}} \leftarrow$ Construct policy tree parameterized by $\boldsymbol{\theta}$
3: $M \leftarrow$ Depth of $\mathcal{T}_{\boldsymbol{\theta}}$
4: $\nu \leftarrow$ Root node of $\mathcal{T}_{\boldsymbol{\theta}}$
5: **for** $l = 1$ to L **do**
6: isTerminal $\leftarrow$ False
7: $s \sim b$
8: **for** $m = 1$ to M **do**
9: $a \leftarrow \theta_{(\nu)}$
10: **if** $a = \varnothing$ **then**
11: // Sample a from the marginal distribution at node n.
12: $a \sim \mathcal{N}\left(\mu_{(\nu)}, \text{diag}(\sigma^2_{(\nu)})\right)$
13: $\theta_{(\nu)} \leftarrow a$
14: **end if**
15: $(s', o, r_m) \leftarrow G(s, a)$
16: $\nu \leftarrow$ Child node of ν via observation edge o
17: $s \leftarrow s'$
18: **if** s is terminal **then**
19: isTerminal $\leftarrow$ True
20: **break**
21: **end if**
22: **end for**
23: **if** isTerminal = False **then**
24: $r_{M+1} \leftarrow$ Heuristic(s)
25: **else**
26: $r_{M+1} \leftarrow 0$
27: **end if**
28: $R_l \leftarrow \sum_{m=1}^{M+1} \gamma^{m-1} r_m$
29: **end for**
30: $V \leftarrow \frac{1}{L} \sum_{l=1}^{L} R_l$
31: **return** $(\boldsymbol{\theta}, V)$

on the entries of the elite parameter vectors in $\mathcal{K}$ that are not $\emptyset$. That is, for the parameter dimension i, we compute the marginal distribution parameters according to $\tilde{\mu}_i = \frac{1}{N_i} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}} \theta_i$ and $\tilde{\sigma}_i^2 = \frac{1}{N_i} \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}} (\theta_i - \tilde{\mu}_i)^2$ respectively (lines 5 and 6), where $N_i = \sum_{\boldsymbol{\theta} \in \mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}}$ (line 3) is the number of parameter vector entries along the i-th dimension that are not $\emptyset$, and $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. Note that we only update the marginal distribution parameters along the i-th dimension if $N_i > 0$ (line 4). Similarly to the basic version of the distribution update in Algorithm 3, we compute the final marginal distribution parameters according to $\mu_i \leftarrow (1 - \alpha)\mu_i + \alpha \tilde{\mu}_i$ and $\sigma_i^2 \leftarrow (1 - \alpha)\sigma_i^2 + \alpha \tilde{\sigma}_i^2$ respectively (lines 7 and 8).

Algorithm 5 UpdateDistribution(Distribution parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, elite samples $\mathcal{K}$, smoothing parameter α)

// The term $D(1-|\mathcal{O}|^{M+1})/(1-|\mathcal{O}|)$ is the size of a parameter vector in $\mathcal{K}$.
for $i = 1$ to $D(1-|\mathcal{O}|^{M+1})/(1-|\mathcal{O}|)$ **do**
  $N_i \leftarrow \sum_{\boldsymbol{\theta}\in\mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}}$
  if $N_i > 0$ **then**
    $\tilde{\mu}_i \leftarrow \frac{1}{N_i} \sum_{\boldsymbol{\theta}\in\mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}} \theta_i$
    $\tilde{\sigma}_i^2 \leftarrow \frac{1}{N_i} \sum_{\boldsymbol{\theta}\in\mathcal{K}} \mathbf{1}_{\{\theta_i \neq \emptyset\}} (\theta_i - \tilde{\mu}_i)^2$
    $\mu_i \leftarrow (1-\alpha)\mu_i + \alpha\tilde{\mu}_i$
    $\sigma_i^2 \leftarrow (1-\alpha)\sigma_i^2 + \alpha\tilde{\sigma}_i^2$
  end if
end for
return $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

While the lazy algorithm is designed to speed up the basic algorithm, they usually do not compute identical results, even when the random number generators used in the algorithms are identical. One reason is that the two algorithms generally use different random numbers for sampling the same quantity, because the lazy algorithm samples policy parameters as needed during policy evaluation, while the basic algorithm samples all policy parameters before policy evaluation. A more important reason is that the lazy update in Algorithm 5 only uses parameters used for the policy evaluation to update the policy distribution, while the basic distribution update in Algorithm 3 uses all policy parameters, even when some of them are not used in policy evaluation. Thus the lazy update generally uses fewer sampled parameters to update the policy distribution, resulting in larger variances for the estimated policy distribution parameters. Thus before convergence, the lazy algorithm has a higher probability to obtain larger updates or perform more aggressive exploration over the policy space. However, empirical results presented in Section 4 indicate that this different exploration behaviour of the lazy algorithm has a minor effect on its performance in terms of the quality of the resulting policies.

Although the lazy update algorithm computes the updated policy distribution using partial policies, it can still be derived from the standard cross-entropy framework described in Section 2.3: as in the basic case, we fit a multivariate normal distribution with diagonal covariance matrix on the policy space; the difference is just that now the partial policies are assumed to be distributed according to the marginal distributions of the multivariate normal distribution. It is easy to show that the maximum likelihood estimates are given by the formulas in Algorithm 5.

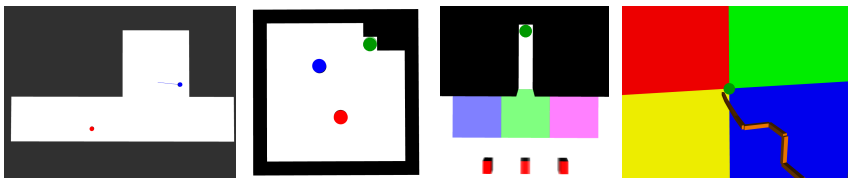
Using the above lazy parameter sampling method can lead to significant computational savings, since only the components of the parameter vectors are sampled that are relevant for evaluating the corresponding policy tree. In our experiments, we investigate the amount of computational savings of our

lazy sampling and evaluation method compared to the basic version discussed in the previous section.

4 Experiments and Results

We tested LCEOPT on 4 decision making problems under partial observability. The problem scenarios are detailed in Section 4.1. Section 4.2 details the experimental setup, while the results are discussed in Section 4.3.

4.1 Problem Scenarios



(a) (b) (c) (d)

Fig. 2 Illustrations of (a) the ContTag, (b) the Pushbox2D, (c) the Parking2D and (d) the SensorPlacement-8 problems. The goal regions in the Pushbox2D, Parking2D and SensorPlacement-8 problems are marked as green circles. Images (b), (c) and (d) are taken from Hoerger et al. (2023).

4.1.1 ContTag

ContTag (Seiler et al., 2015) is a modified version of the popular POMDP benchmark problem Tag (Pineau et al., 2003). An agent operates in a 2D-environment (shown in Figure 2(a)) where it has to tag an opponent, while the opponent is actively trying to avoid the agent. The state space is a five-dimensional continuous space consisting of the location (x_r, y_r) and orientation ϕ_r (expressed in radians) of the agent and the location (x_o, y_o) of the opponent. The action space is $\mathcal{A} = [-\pi, \pi] \cup \{\text{TAG}\}$, where the first component is the set of all angular directions the agent can move towards, whereas the second component is an additional tag action. At each step, in case the agent executes a directional action, its orientation and position evolve deterministically according to $\phi'_r = \phi_r + a$, $x'_r = x_r \cos(\phi'_r)$ and $y'_r = y_r + \sin(\phi'_r)$. Simultaneously, the opponent attempts to move away from the agent and its next location (x'_o, y'_o) is computed according to $x'_o = x_o + \cos(\phi) + e_x$ and $y'_o = y_o + \sin(\phi) + e_y$, where $\phi = \text{atan2}(y_o - y_a, x_o - x_a)$ is the angle between the agent and the robot, and e_x and e_y are random motion errors drawn from a truncated Normal distribution $\mathcal{N}(\mu, \sigma^2, l, u)$, which is the Normal distribution $\mathcal{N}(\mu, \sigma^2)$ truncated to the interval $[l, u]$. For our experiments, we set $\mu = 0$, $\sigma = \frac{\pi}{8}$, $l = -\frac{\pi}{8}$ and $u = \frac{\pi}{8}$. In case the agent's or the opponent's next state would collide with the boundary region, their positions remain the same. If the agent executes the TAG action, its position and orientation remains unchanged as well.

The initial positions of the agent and the opponent are drawn from a uniform distribution over the free space of the environment, while the initial orientation of the agent is set to 0. While the agent knows its initial position and orientation, the position of the opponent is unknown. However, the agent has access to a noisy sensor with outputs $\{\text{DETECTED}, \text{NOT DETECTED}\}$ to detect the opponent. If the opponent is visible, i.e., the relative angle $\phi - \phi_r$ between the agent and the opponent is inside the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, the sensor produces the output DETECTED with probability $p = 1 - \frac{\phi - \phi_r}{\pi}$ and NOT DETECTED with probability $1 - p$. Otherwise, the sensor deterministically produces NOT DETECTED.

Upon activating the TAG action, the agent receives a reward of 10 if its Euclidean distance to the agent is smaller than one unit length. Otherwise the agent receives a penalty of -10. Every other action incurs a small penalty of -1. The problem terminates if the opponent is successfully tagged, or a maximum of 90 steps has been reached. The discount factor is 0.95.

Note that the action space in this problem is a hybrid space, consisting of both continuous and discrete variables. For LCEOPT, we embed the action space into the two-dimensional interval $[-\pi, \pi] \times [-1, 1]$ and define that the agent executes the TAG action, if the second component of the action is in the interval $[0, 1]$.

4.1.2 Pushbox

Pushbox (Seiler et al., 2015) is a scalable motion planning problem, based on air hockey. A disk-shaped robot (shown as a blue disk in Figure 2(b)) has to push a disk-shaped puck (shown as a red disk in Figure 2(b)) into a green circle goal region (shown as a green circle in Figure 2(b)), while avoiding any collisions with a boundary region (shown as a black region in Figure 2(b)). If the puck is successfully pushed into the goal region, the robot receives a reward of $1,000$, but if either the robot or the puck collides with the boundary region, the robot receives a penalty of $-1,000$. Additionally, the robot incurs a penalty of -10 for every step taken. The robot can move around the environment by selecting a displacement vector. Upon colliding with the puck, the puck is pushed away, and its motion is affected by noise. The initial position of the robot is known, but the initial puck position is uncertain. However, the robot has access to a noisy bearing sensor to localize the puck. Additionally, the robot receives a binary observation from a contact sensor, indicating if a contact between the robot and the puck occurred. We consider two variants of the problem: **Pushbox2D** and **Pushbox3D**, which differ in the dimensionality of the state and action spaces. The former (illustrated in Figure 2(a)) operates in a 2D plane, while the latter operates inside a 3D environment. More details on the Pushbox problem can be found in Seiler et al. (2015).

4.1.3 Parking

The Parking problem, proposed in Hoerger et al. (2023) and shown in Figure 2(c), is a navigation problem in which a vehicle with deterministic

dynamics operates in an environment populated by obstacles. The vehicle's goal is to safely reach a specified goal location while avoiding collisions with the obstacles. Reaching the goal earns a reward of 100, while collisions with obstacles incur a penalty of -100, and every step taken incurs a penalty of -1. The vehicle starts near one of three possible starting locations (red locations in Figure 2(c)) with equal probability. There are three distinct areas in the environment with different types of terrain (colored areas in Figure 2(c)), and the vehicle receives observations about the terrain type upon traversal. The observations are only correct 70% of the time due to sensor noise. Here we consider two variants of the problem, **Parking2D** and **Parking3D**, with different state and action spaces. In Parking2D, the state consists of the vehicle's position, orientation, and velocity on a 2D plane, and the action space consists of the steering wheel angle and acceleration. In Parking3D, the vehicle operates in full 3D space, and the state and action spaces have additional components to model the vehicle's elevation and change in elevation, respectively. The problem is challenging due to multi-modal beliefs and the narrow passage to the goal, which makes good rewards scarce. Additional details can be found in Hoerger et al. (2023).

4.1.4 SensorPlacement

SensorPlacement, proposed in Hoerger et al. (2023) and shown in Figure 2(d), is a scalable motion planning under uncertainty problem, where a manipulator with D degrees of freedom (DOF) and D revolute joints operates inside a 3D environment with muddy water. The manipulator is situated in front of a marine structure, consisting of four walls (colored walls in Figure 2(d)), and its task is to attach a sensor at a specific goal area located between the walls, which is reward by $1,000$, while avoiding collisions with the walls, which is penalized by -500. Additionally, the robot receives a penalty of -1 for every step. The state space consists of the joint angles for each joint, and the action space is a set of joint velocities. Initially, the robot is uncertain about its joint angle configuration and, due to underwater currents, the robot is subject to random control errors. To localize itself, the manipulator's end-effector is equipped with a touch sensor which provides noise-free information about the wall being touched. The problem has four variants, denoted as SensorPlacement-D, with D ranging from 6 to 12, which differ in the number of revolute joints and the dimensionality of the action space. The discount factor is $\gamma = 0.95$, and the robot must mount the sensor within 50 steps while avoiding collisions with the walls to succeed. Additional details can be found in (Hoerger et al., 2023).

4.2 Experimental Setup

The purpose of our experiments is two-fold: First is to compare LCEOPT with three state-of-the-art online POMDP solvers for continuous action spaces, POMCPOW (Sunberg & Kochenderfer, 2018), VOMCPOW (Lim et al., 2021) and ADVT (Hoerger et al., 2023) on the above problem scenarios. To do

this, we implemented LCEOPT using the lazy parameter sampling method described in Section 3.4.2, the tree baseline solvers and the problem scenarios in C++ using the OPPT framework (Hoerger, Kurniawati, & Elfes, 2018). To approximately determine the best parameters for each solver in the problem scenarios, we used the CE-method for optimization over the solver's parameter spaces. Apart from solver-intrinsic parameters, the parameter space also includes the effective planning horizon, i.e., the maximum depth of the lookahead trees for POMCPOW, VOMCPOW and ADVT, and the maximum depth of the policy trees for LCEOPT. 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 parameters for each solver and their parameter ranges are detailed in Section A.

Second is to understand the computational benefits of our proposed lazy parameter sampling, evaluation, and distribution update method described in Section 3.4.2 compared to the basic method described in Section 3.4.1. 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 5), 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 a AMD EPYC 7003 CPU with 4GB of memory. The next section discusses the results of our experiments.

4.3 Results

This section presents the results of our experiments. Section 4.3.1 presents the comparison with state-of-the-art methods, while Section 4.3.2 discusses the effects of our lazy sampling method on the computational efficiency of LCEOPT.

4.3.1 Comparison with State-of-the-Art Methods

Table 1 shows the average total discounted rewards of all tested solvers for the ContTag, Pushbox and Parking problems. The results for the SensorPlacement problems are shown in Table 2. It can be seen that LCEOPT outperforms the baseline solvers in all problems, except for the ContTag problem, in which ADVT performs slightly better.

Table 1 Average total discounted rewards and 95% confidence intervals of all tested solvers for the ContTag, Pushbox and Parking 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.

	ContTag	Pushbox2D	Pushbox3D	Parking2D	Parking3D
LCEOPT (Ours)	0.02 ± 0.23	**399.7 ± 8.7**	**358.6 ± 12.3**	**53.4 ± 0.4**	**47.2 ± 0.6**
ADVT	**0.37 ± 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

Table 2 Average total discounted rewards and 95% confidence intervals of all tested solvers for the SensorPlacement problems. The average is taken over 1,000 simulation runs per solver and problem, with a planning time of 1s per step.

	SensorPlacement-6	SensorPlacement-8	SensorPlacement-10	SensorPlacement-12
LCEOPT (Ours)	**914.3 ± 2.6**	**885.5 ± 2.9**	**858.8 ± 4.2**	**832.1 ± 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

A notable difference in performance between LCEOPT and the baselines arises in the SensorPlacement problems. The results Table 2 indicate that LCEOPT scales substantially better as we increase the dimensionality of the action space. 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 when looking at the results of the Parking problems in Table 1, where the performance of LCEOPT suffers only marginally compared to the baseline solvers as we increase the dimensionality of the action space. 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 instead works with distributions over the policy space. At each node in the policy trees, LCEOPT maintains and updates a sampling distribution that quickly focuses it's 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 insight regarding the results is that all solvers require only a relatively short planning horizon for most of the problem scenarios. Table 3 shows the effective planning horizons for all solvers for the ContTag, Pushbox, Parking and SensorPlacement problems, as found during the parameter-tuning process. For the ContTag, Pushbox and SensorPlacement problems, all solvers require an effective planning horizon of only two steps. The reason is the heuristic estimate of $V^*(b)$ described in Section 3.4.1 used

Table 3 Effective planning horizons of all solvers in the problem scenarios. For POMCPOW, VOMCPOW and ADVT, the effective planning horizon refers to the maximum depth of the lookahead trees constructed during planning, while for LCEOPT, it refers to the maximum depth of the policy trees.

	ContTag	Pushbox2D/3D	Parking2D/3D	SensorPlacement-D
LCEOPT	2	2	5	2
ADVT	2	2	5	2
VOMCPOW	2	2	5	2
POMCPOW	2	2	5	2

to evaluate leaf nodes of the policy trees for LCEOPT and the lookahead trees for POMCPOW, VOMCPOW and ADVT respectively. For all problem scenarios, we designed simple state-dependent heuristic estimates of $V^*(b)$, that assume that the problem is deterministic. 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. The reason is that in this problem, 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 are often difficult to capture via simple state-dependent heuristics, leading to a longer effective planning horizon required to find good solutions.

4.3.2 Comparison of the basic Method and the Lazy Method

Table 4 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 shallow policy trees (up to $M = 4$), while the lazy method performs slightly better for policy trees of depth $M = 5$. However, the results for the SensorPlacement-12 problem indicate that the lazy method outperforms the basic one significantly in this problem, even for more shallow policy trees. The reason is that the dimensionality of the parameter space increases dramatically for deeper policy trees, due to the 12-dimensional action 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 5 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

Table 4 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 $L = 11$.

		$M = 1$	$M = 2$	$M = 3$	$M = 4$	$M = 5$
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 5 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.

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

behaviours as discussed in Section 3.4.2, 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.

5 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 POMDPs with continuous action spaces 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.

Acknowledgements. We thank Jerzy Filar for many helpful discussions. This work is partially supported by the Australian Research Council (ARC) Discovery Project 200101049.

Appendix A Solver Parameters

Table A1 shows the parameter ranges used when searching for the best parameter of each solver. For all problem scenarios, we use the same parameter ranges. For LCEOPT, the parameters N, L, K, M, α and $\boldsymbol{\sigma}^2_{\text{init}}$ refer to the number of candidate policies per iteration, number of sampled trajectories per policy, number of elite samples, policy tree depth, smoothing factor and the variance of the initial distribution respectively. In all our experiments we set $\boldsymbol{\mu}_{\text{init}} = \mathbf{0}$ and $\boldsymbol{\sigma}^2_{\text{init}} = \sigma^2_{\text{init}}\mathbf{1}$, where $\mathbf{0}$ and $\mathbf{1}$ are vectors of ones and zeroes respectively. Details regarding the parameters for ADVT can be found in Hoerger et al. (2023), while details regarding the parameters for VOMCPOW and POMCPOW can be found in Lim et al. (2021). To find the best set of parameters for each solver and problem scenario, we apply the CE-method for 100 iterations, using a multivariate Gaussian distribution with diagonal covariance matrices (similarly to LCEOPT). The best parameter is then chosen to be the mean of the resulting distribution over the parameter space.

Table A1 Solver parameter and parameter ranges used when searching for the best parameters for all tested solvers in each problem scenario.

	N	L	K	M	α	σ^2_{init}
LCEOPT	$[10, 100]$	$[1, 500]$	$[1, 500]$	$[1, 10]$	$[0, 1]$	$[0.01, 4.0]$
	C	L	C_r			
ADVT	$[2, 500]$	$[1, 500]$	$[0.1, 100]$			
	c	k_a	α_a	k_o	α_o	ω
VOMCPOW	$[2, 1]$	$[1, 50]$	$[0.001, 5]$	$[1, 50]$	$[0.001, 5]$	$[0, 1]$
POMCPOW	$[2, 1]$	$[1, 50]$	$[0.001, 5]$	$[1, 50]$	$[0.001, 5]$	–

References

Agha-mohammadi, A.-a., Chakravorty, S., Amato, N.M. (2011). Firm: Feedback controller-based information-state roadmap - a framework for motion planning under uncertainty. *2011 IEEE/RSJ international conference on intelligent robots and systems* (pp. 4284–4291).

Arulampalam, M., Maskell, S., Gordon, N., Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, *50*(2), 174–188, https://doi.org/10.1109/78.978374

Bai, H., Hsu, D., 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, https://doi.org/10.1177/0278364914528255

Botev, Z.I., Kroese, D.P., Rubinstein, R.Y., L'Ecuyer, P. (2013). The cross-entropy method for optimization. C. Rao & V. Govindaraju (Eds.), *Handbook of statistics - machine learning: Theory and applications* (pp. 35–59). Elsevier.

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., Bonnard, N. (2011). Continuous upper confidence trees. *Proc. learning and intelligent optimization* (pp. 433–445). Springer.

Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. H.J. van den Herik, P. Ciancarini, & H.H.L.M.J. Donkers (Eds.), *Computers and games* (pp. 72–83). Springer.

de Boer, P.-T., Kroese, D.P., Mannor, S., Rubinstein, R.Y. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, *134*(1), 19–67, https://doi.org/10.1007/s10479-005-5724-z

Filar, J.A., Qiao, Z., Ye, N. (2019). POMDPs for sustainable fishery management. *23rd international congress on modelling and simulation-supporting evidence-based decision making: The role of modelling and simulation, MODSIM 2019* (pp. 645–651).

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J. (2019). Learning latent dynamics for planning from pixels. *Proc. of the 36th international conference on machine learning* (Vol. 97, pp. 2555–2565). PMLR.

Hoerger, M., Kurniawati, H., Bandyopadhyay, T., Elfes, A. (2020). Linearization in motion planning under uncertainty. K. Goldberg, P. Abbeel, K. Bekris, & L. Miller (Eds.), *Algorithmic foundations of robotics XII: Proceedings of the twelfth workshop on the algorithmic foundations of robotics* (pp. 272–287). Springer.

Hoerger, M., Kurniawati, H., Elfes, A. (2018). A software framework for planning under partial observability. *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (p. 1-9).

Hoerger, M., Kurniawati, H., Kroese, D., Ye, N. (2023). Adaptive discretization using voronoi trees for continuous-action POMDPs. S.M. LaValle, J.M. O'Kane, M. Otte, D. Sadigh, & P. Tokekar (Eds.), *Algorithmic foundations of robotics XV: Proceedings of the twelfth workshop on the algorithmic foundations of robotics* (pp. 170–187). Springer.

Kaelbling, L.P., Littman, M.L., Cassandra, A.R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1-2), 99–134, https://doi.org/10.1016/s0004-3702(98)00023-x

Kim, B., Lee, K., Lim, S., Kaelbling, L., Lozano-Pérez, T. (2020). Monte Carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. *Proceedings of the AAAI conference on artificial intelligence* (Vol. 34, pp. 9916–9924). AAAI.

Kurniawati, H. (2022). Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*, 253–277, https://doi.org/10.1146/annurev-control-042920-092451

Kurniawati, H., Du, Y., Hsu, D., 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, https://doi.org/10.1177/0278364910386986

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

Kurniawati, H., & Yadav, V. (2016). An online POMDP solver for uncertainty planning in dynamic environment. *Robotics research: The 16th international symposium ISRR* (pp. 611–629). Springer.

Lim, M.H., Tomlin, C.J., Sunberg, Z.N. (2021). Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation POMDPs. *2021 60th IEEE conference on decision and control (CDC)* (pp. 4493–4500).

Lindquist, A. (1973). On feedback control of linear stochastic systems. *SIAM Journal on Control*, *11*(2), 323–343, https://doi.org/10.1137/0311025

Mannor, S., Rubinstein, R.Y., Gat, Y. (2003). The cross entropy method for fast policy search. *Proceedings of the 20th international conference on machine learning* (pp. 512–519). AAAI Press.

Mern, J., Yildiz, A., Sunberg, Z., Mukerji, T., Kochenderfer, M.J. (2021). Bayesian optimized Monte Carlo planning. *Proceedings of the AAAI conference on artificial intelligence* (Vol. 35, pp. 11880–11887). AAAI.

Oliehoek, F.A., Kooij, J.F.P., Vlassis, N. (2008). A cross-entropy approach to solving Dec-POMDPs. C. Badica & M. Paprzycki (Eds.), *Advances in intelligent and distributed computing* (pp. 145–154). Springer.

Omidshafiei, S., Agha-mohammadi, A.-a., Amato, C., Liu, S.-Y., How, J.P., Vian, J. (2016). Graph-based cross entropy method for solving multi-robot decentralized POMDPs. *2016 IEEE international conference on robotics and automation (icra)* (pp. 5395–5402).

Papadimitriou, C.H., & Tsitsiklis, J.N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450, https://doi.org/10.1287/moor.12.3.441

Pineau, J., Gordon, G., Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. *Proceedings of 18th international joint conference on artificial intelligence (IJCAI '03)* (pp. 1025–1032).

Rubinstein, R.Y., & Kroese, D.P. (2004). *The cross entropy method: A unified approach to combinatorial optimization, Monte Carlo simulation and machine learning*. Springer.

Schwartz, J., Kurniawati, H., El-Mahassni, E. (2020). POMDP+ information-decay: Incorporating defender's behaviour in autonomous penetration testing. *Proceedings of the international conference on automated planning and scheduling* (Vol. 30, pp. 235–243).

Seiler, K.M., Kurniawati, H., Singh, S.P. (2015). An online and approximate solver for POMDPs with continuous action space. *2015 IEEE international conference on robotics and automation* (pp. 2290–2297).

Silver, D., & Veness, J. (2010). Monte-carlo planning in large POMDPs. J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems* (Vol. 23). Curran Associates.

Smith, T., & Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. *Proceedings of the twenty-first conference on uncertainty in artificial intelligence* (pp. 542–549). AUAI Press.

Sun, W., Patil, S., Alterovitz, R. (2015). High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics*, *31*(1), 104–116, https://doi.org/10.1109/tro.2014.2380273

Sunberg, Z., & Kochenderfer, M. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. *Proceedings of the international conference on automated planning and scheduling* (Vol. 28, pp. 259–263). AAAI Press.

van den Berg, J., Abbeel, P., 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, https://doi.org/10.1177/0278364911406562

van den Berg, J., Patil, S., Alterovitz, R. (2012). Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, *31*(11), 1263–1278, https://doi.org/10.1177/0278364912456319

Wang, E., Kurniawati, H., Kroese, D. (2018, Jun.). An on-line planner for POMDPs with large discrete action space: A quantile-based approach. *Proceedings of the international conference on automated planning and scheduling* (Vol. 28, pp. 273–277). AAAI Press.

Ye, N., Somani, A., Hsu, D., Lee, W.S. (2017). DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research*, *58*, 231–266, https://doi.org/10.1613/jair.5328