Optimization and Engineering
https://doi.org/10.1007/s11081-024-09896-2

RESEARCH ARTICLE

Online model adaptation in Monte Carlo tree search planning

Maddalena Zuccotto[1] · **Edoardo Fusa**[1] · **Alberto Castellini**[1] · **Alessandro Farinelli**[1]

Received: 30 November 2023 / Revised: 1 December 2023 / Accepted: 1 June 2024

Abstract
We propose a model-based reinforcement learning method using Monte Carlo Tree Search planning. The approach assumes a black-box approximated model of the environment developed by an expert using any kind of modeling framework and it improves the model as new information from the environment is collected. This is crucial in real-world applications, since having a complete knowledge of complex environments is impractical. The expert's model is first translated into a neural network and then it is updated periodically using data, i.e., state-action-next-state triplets, collected from the real environment. We propose three different methods to integrate data acquired from the environment with prior knowledge provided by the expert and we evaluate our approach on a domain concerning air quality and thermal comfort control in smart buildings. We compare the three proposed versions with standard Monte Carlo Tree Search planning using the expert's model (without adaptation), Proximal Policy Optimization (a popular model-free DRL approach) and Stochastic Lower Bounds Optimization (a popular model-based DRL approach). Results show that our approach achieves the best results, outperforming all analyzed competitors.

Keywords Model based reinforcement learning · Learning dynamics models · Monte Carlo tree search · Planning and learning · Adaptive systems

Mathematics Subject Classification 68T05 · 60J05

✉ Maddalena Zuccotto
maddalena.zuccotto@univr.it

Edoardo Fusa
edoardo.fusa@gmail.com

Alberto Castellini
alberto.castellini@univr.it

Alessandro Farinelli
alessandro.farinelli@univr.it

[1] Department of Computer Science, University of Verona, Strada Le Grazie 15, Verona 37134, Italy

Published online: 18 June 2024

1 Introduction

Reinforcement Learning (RL) has recently proved to have strong potentials in applications where agents must operate in unknown environments adapting to unexpected (or partially specified) situations. The goal of RL algorithms is to learn optimal policies, namely, functions that map system states to actions that allow the agent to reach its goal. Model-free RL focuses, in particular, on algorithms that learn the policy without making any assumption about the dynamics of the environment. These methods directly learn the policy from relationships among states, actions and rewards observed in the environment while the agent acts. On the other hand, model-based RL leverages interactions with the environment to learn a model that is used in the computation of the policy. This approach provides a richer formalism allowing to integrate prior knowledge about the environment, hence achieving better performance, however, it requires information about the model of the dynamics (e.g., the form of that model), often unavailable in real-world applications.

In this work, we aim at addressing this limitation by proposing a model-based RL approach that improves an expert-defined approximate model of the dynamics according to the information gathered online from the environment. The technique employs Artificial Neural Networks (ANN) (Goodfellow et al. 2016) to represent the model of the dynamics and Monte Carlo Tree Search (MCTS) (Browne et al. 2012) as an internal planner using the ANN to perform simulations. The approach first transforms the approximate model provided by the expert into a neural network. In this way it allows the usage of any kind of expert-defined model of the dynamics (e.g., rule-based, probability-based, data-driven, etc.). Then, it uses the ANN model as a simulator in MCTS to compute action Q-values in the visited states. Third, it performs optimal actions (i.e., optimal according to the current model of the environment and the number of simulations used by MCTS) in the real environment (which is typically different from the current model) and collects the observed next states and reward signals. Finally, it uses information about next states and rewards to update the ANN model of the dynamics, which will be used by MCTS to compute optimal actions in the following steps.

The online nature of MCTS well suits the online update of the ANN model. The choice of MCTS as a planner is related to its capability to compute the Q-function online and locally (i.e., only for the states actually visited by the agent), which allows scaling to very large domains typical of real-world applications. Furthermore, MCTS sampling does not require a full transition matrix, as needed by several RL methods, but it only requires a black-box simulator, namely, a function providing the next state given the current state-action pair (notice that this function can be also stochastic). Thus, the expert is assumed to provide only such a function, which is simpler than providing a transition matrix with probabilities for all triplets *(state, action, next-state)*.

A key point of our approach is related to how it integrates new information in the current model. We propose a method for solving this problem and three ways to perform the integration. The first version, *MCP_Real*, simply updates periodically the weights of the copy neural network using environment data as a training set. The second version, *MCP_Mix*, merges data (i.e., triplets state-action-next-state) generated by the original expert's model with data from the environment and periodically re-

trains the copy neural network using this dataset as a training set. The third version, *MCP_Select*, keeps the copy neural network of the expert's model unchanged and trains a separate neural network with data from the environment, then during Monte Carlo simulations it selects the best model to perform each step according to how close is the current state-action pair to the training set by which the model has been trained.

To evaluate our approach we consider a real-world application related to air quality and thermal comfort control in smart buildings. In our domain (Bianchi et al. 2023; Capuzzo et al. 2022) the planner acquires online information about internal temperature, external temperature, CO_2 level, Volatile Organic Compounds (VOC) level, and people occupancy, and suggests at each step the best action to perform among activation/deactivation of ventilation, activation/deactivation of sanitization, opening/closing windows. These actions should consider future presence in the room and future evolution of external temperature (which are considered available throughout the current day) hence the planning component is crucial to achieve good performance. We evaluated the performance of the three versions of our method and compared them to three baseline approaches, namely, (i) MCTS using the expert's model with no adaptations to the real environment; (ii) Proximal Policy Optimization (PPO), a popular model-free approach (Schulman et al. 2017) representing the state-of-the-art among model-free approaches; (iii) Stochastic Lower Bounds Optimization (SLBO), a recent and popular model-based approach (Luo et al. 2019). The second version of our approach, called MCP_M in the following, achieved the best performance among the proposed versions. More important, this version outperformed the three state-of-the-art competitors, in terms of discounted return, in our reference domain.

In summary, the contributions of this work are threefold:

- we propose an adaptive model-based approach to compute and improve a policy considering an approximated model of the environment. The approach is based on MCTS for planning and on ANN for model adaptation.
- We apply our approach to a real-world domain concerning air quality and thermal comfort control in an indoor environment.
- We successfully compare our approach with state-of-the-art model-based and model-free RL methods showing that our approach outperforms state-of-the-art methods in terms of return and sample efficiency in a domain concerning air quality control in indoor environments.

2 Related work

Our work has connections with model-based RL, MCTS-based planning and continual learning. Here, we analyze the shared features and highlight the original elements of the proposed method.

2.1 Model-based RL

RL algorithms (Sutton and Barto 2018) can be categorized in model-free (Kaelbling et al. 1996) and model-based (Moerland et al. 2023; Luo et al. 2022). Model-free

methods have been significantly improved in the last decade achieving very good performance where two of the most prominent approaches are Trust Region Policy Optimization (TRPO) (Schulman et al. 2015) and Proximal Policy Optimization (PPO) (Schulman et al. 2017). These algorithms can be applied to domains in which the model of the environment (i.e., transition and reward models) are completely unknown, which is very useful in some contexts. In several real-world applications, however, the environment is known or partially known and using a model of the environment can significantly improve sample efficiency of RL approaches (i.e., reduce the number of interactions with the environment) hence yielding better policies.

In the landscape of model-based RL methods, a well known approach is Dyna (Sutton and Barto 2018; Sutton 1991, 1990) which however uses tabular Q-learning that has scaling problems and cannot be applied to real-world domains with large and continuous state and action spaces. Dyna-Style algorithms (Luo et al. 2019; Clavera et al. 2018; Kurutach et al. 2018), improve the policy by using rich imagined data (Wang et al. 2019). Contrarily, policy search with backpropagation trough time algorithms (Chebotar et al. 2017; Finn et al. 2016; Deisenroth et al. 2015; Heess et al. 2015; Tassa et al. 2012; Deisenroth and Rasmussen 2011) leverage the model derivatives to improve the policy (Wang et al. 2019). Model-based shooting algorithms (Chua et al. 2018; Nagabandi et al. 2018; Rao 2009; Richards 2005), approximately solve horizon problems of Model Predictive Control (MPC) in the case of non-linear dynamics and non-convex reward functions (Wang et al. 2019). In the context of dynamical systems the problem of model learning (i.e., the inverse problem) has been investigated from several perspectives and using different modeling frameworks. In case the system dynamics is represented by a set of differential equations, most established approaches rely, for instance, on deep neural networks (Raissi 2018) and gaussian mixture models (Khansari-Zadeh and Billard 2011). When a more complex mathematical formalization for the system dynamics is required, e.g. cellular automata (Dennunzio et al. 2023, 2019), graph neural architecture can be used to learn the corresponding model (Grattarola et al. 2021). An important difference between these state-of-the-art model-based RL methods and the approach that we propose is that our method uses MCTS to compute the policy online. Moreover, we aim to improve the model of the dynamics leveraging both approximated data, used as prior knowledge about the model, and real-world observations.

2.2 MCTS planning

MCTS planning (Browne et al. 2012) and UCT (Kocsis and Szepesvári 2006) are approaches that compute the policy online and locally (i.e., only for the states actually visited by the agent). Hence they allow to mitigate the scaling problem using sampling based strategies. A recent popular result obtained with these approaches is that of AlphaGo (Silver et al. 2016, 2017), a program which defeated a world champion in the game of Go. There exists also extensions of MCTS-based planning to partially observable environments. A key work in this area proposes Partially Observable Monte-Carlo Planning (Silver and Veness 2010), a technique for scaling planning to large environments. However, the technique assumes a known model of the environment. Some

methods have been proposed to learn different elements of the environment during the agent-environment interaction. For instance, in (Zuccotto et al. 2023, 2022; Castellini et al. 2021, 2019) some state-variable relationships are learned to improve belief initialization and update, in (Mazzi et al. 2023a, b, 2021) logic specifications are learned for introducing soft policy guidance and shielding in POMCP, and in (Giuliari et al. 2021; Wang et al. 2020) visual elements of the unknown environments are learned for solving the problem of Active Visual Search. In the context of Bayesian adaptive learning in POMDPs (Ross et al. 2011), Katt et al. (2017) present an elegant method to learn the transition and reward models in which authors extend the POMCP algorithm to the Bayes-Adaptive case, proposing the Bayes-Adaptive Partially Observable Monte Carlo Planning (BA-POMCP) approach that, however, learns the parameters of the transition model. However, to the best of our knowledge the only attempt to perform model learning in MCTS-based planning on observable environment is that of Bayesian Adaptive Monte Carlo Planning (Guez et al. 2013). This approach performs Bayesian learning that leads to a significant, often prohibitive, computational complexity since it considers all possible models according to a belief (i.e., a probability distribution) over their parameters. The technique proposed in this paper instead performs standard learning based on a neural network model, providing a simple and sample efficient alternative. Finally, a methodology for safely improving a baseline policy using MCTS has been recently proposed (Castellini et al. 2023). It assumes the transition model to be initially unknown, while the baseline policy and a dataset of trajectories to be available. The approach learns transitions probabilities from the dataset, which is collected using the baseline policy, but the learning phase is performed offline and the model has a tabular form.

2.3 Continual learning

Continual Learning (CL) (Lesort et al. 2020) shares some similar concepts with our work since training data are not available at once and need to be integrated by keeping the most representative data, as rehearsal approaches do (Lesort et al. 2019; Rebuffi et al. 2017). However, CL algorithms aim at learning new tasks preserving the knowledge of previously learned tasks, considering that old tasks were correctly learned. Our technique instead starts with an expert's model of the environment which is approximated and partial, and it updates the model in the parts that do not fit the observations taken from the real environment. Furthermore, in our approach model adaptation/update is inserted in a process of policy improvement typical of RL methods.

3 Background

We introduce the main tools used in this work, namely, Markov Decision Process (MDP), MCTS-based planning and artificial neural networks.

3.1 MDPs

An MDP (Russell and Norvig 2010) is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ is a finite set of $states$, $\mathcal{A}$ is a finite set of $actions$, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ is the *transition model* ($\Pi(\mathcal{S})$ space of probability distributions over states), $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the *reward function*, and $\gamma \in [0, 1)$ is the *discount factor*. The agent's goal is to maximize the *expected discounted return*, namely, the sum of weighted rewards over long runs $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)]$. The solution of an MDP is a *policy*, namely, a function that specifies the action the agent should perform in any reachable state. A policy is optimal if it maximizes the expected discounted return.

3.2 MCTS planning

MCTS (Browne et al. 2012) is a popular heuristic search algorithm that recently achieved very important results in board games, such as Go and Chess. It progressively builds the search tree descending from the root node guided by the results of previous descents. The UCT (Kocsis and Szepesvári 2006) strategy is used to address the exploration-exploitation dilemma. The MCTS algorithm repeats a sequence of four steps (Browne et al. 2012): *selection* in which the *tree policy* is recursively applied starting from the root node to descend the tree and determine the best node to expand; *expansion* that expands the selected node with a child chosen among the available actions; *simulation* in which a *default policy* is applied to simulate a run from the expanded node and produce a reward; *backpropagation* that propagates the reward and updates the statistics of the visited nodes up to the root node. The simulation step uses the model of the environment to generate the trajectories on which the reward values, and therefore the policy, are computed. Thus, it is important to use an accurate model of the environment to obtain high accuracy in the computation of the reward values and, as a consequence, of a good policy.

3.3 Artificial neural networks

An Artificial Neural Network (ANN) (Goodfellow et al. 2016) is a computational model that consists of information-processing units, *neurons*, interconnected by links, *synapses*, characterized by a real-valued *weight* representing their strength. Each *node* aggregates its input signals and applies to the resulting one a nonlinear function, the *activation function* (e.g. Sigmoid, ReLU), producing node's output. The activation function parameters are the weights of network's connection. Typically, to learn the weights an ANN uses stochastic gradient methods (e.g., SGD and Adam), that adjust the direction of each weight to improve the overall network performance according to the minimization, or maximization, of an objective function. A typical function in supervised learning is the expected error, used as a *loss* function and computed over labeled training data.

4 Methodology

We introduce the two main steps of our methodology and the three variants that characterize the second step.

4.1 Proposed approach

The methodology here proposed is composed of two main steps (points 1 and 2 in Fig. 1). In the first step, we assume an expert provides a mathematical model to approximate the dynamics of the real environment. This model is a black box function (we do not need to have any knowledge about its implementation) that receives a state and an action and returns the next state. We perform random sampling on the overall state and action spaces, and for each state-action pair we provide it to the model and store the related next state returned by the model. In this way, we generate a dataset $\widehat{\mathcal{D}}$ containing a large amount of triplets state-action-next-state for different environment conditions (i.e., states) and actions. Then, we generate a ANN-copy $\widehat{\mathcal{T}}^{ANN}$ of the approximated model by training the ANN on dataset $\widehat{\mathcal{D}}$. The function approximation capabilities of ANN allow us to work with any type of expert's model. This first step, described in detail in the next section and shown in the horizontal rectangle on top of Fig. 1, is performed only once. In the second step, described in detail in the subsequent section, we use the network $\widehat{\mathcal{T}}^{ANN}$ inside MCTS to perform simulations and we update the network as new observations are collected from the environment. Three versions of this step are proposed (see the three vertical rectangles in Fig. 1) that merge in different ways the prior knowledge in the ANN and the information in the collected data.

4.1.1 Generation of ANN-Copy of the Expert's Model

We use the expert's model $\widehat{\mathcal{T}}$ to generate the dataset $\widehat{\mathcal{D}}$ of simulated data (see the cylinder on top of Fig. 1), i.e., a collection of triplets (s, a, s') where s and a, respectively, represent a state of the environment and an action performed by the agent, and s' is the state of system after performing the action. More precisely, we randomly choose an initial state and run the expert's model performing actions randomly sampled over the action space to generate the (s, a, s') triplets for that day. Then, we train a neural network $\widehat{\mathcal{T}}^{ANN}$ on $\widehat{\mathcal{D}}$ obtaining a copy of the expert's model. Technical details about the architecture of the ANN and the amount of data needed to learn it depends on the properties of the expert's model (examples are given in Sect. 6.1).

4.1.2 Transition model update

The approximated transition model $\widehat{\mathcal{T}}^{ANN}$ can be updated in several ways as the agent acts in the environment and collects information about its dynamics and reward. The three approaches here proposed, named MCP_R, MCP_S and MCP_M, integrate in three different ways the knowledge in the expert's model with the information gathered step-by-step by the agent as it observes true values of state-variables (i.e.,

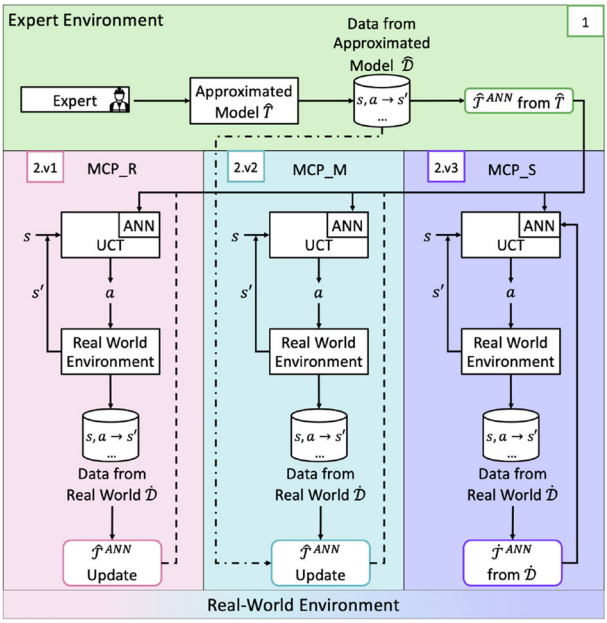

Fig. 1 Two main steps of the proposed methodology. The green horizontal rectangle shows the generation of an approximated transition model (the expert's model); the pink, light blue and purple vertical rectangles show the three versions of the model update procedure

sensor readings in real-world applications). The three model adaptation strategies are explained in the following and a flow chart for each of them is depicted in the bottom of Fig. 1.

$MCP\_Real(MCP\_R)$: *update of* $\widehat{\mathcal{T}}^{ANN}$ *using only data from the environment.* This version stores in a dataset $\dot{\mathcal{D}}$ the triplets $(\dot{s}, \dot{a}, \dot{s}')$ of state, action and next state observed by the agent in the real-world environment. Every n time-steps MCP_R performs a re-training of the neural network $\widehat{\mathcal{T}}^{ANN}$ updating the current weights of the network using data in $\dot{\mathcal{D}}$ as a training set. The updated network is used in MCTS simulations for the next n time-steps (i.e., until the next update). This update strategy has the advantage of being simple but it suffers the risk of catastrophic forgetting since the re-training is based only on data from the environment, hence the knowledge in the expert's model, stored in the weights of the initial network, can be quickly lost as weights are updated. This issue could be particularly harsh when observations from the real-world environment are very concentrated on small parts of the state-action space, as in the first update stages when $\dot{\mathcal{D}}$ contains a few observations. In these cases, the expert's knowledge about unobserved parts of the state-action space could be lost by updating the weights only considering data from the observed part of the space. The update strategies presented in the following aim to mitigate this problem, preserving expert's knowledge in parts of the state-action domains not already visited by the agent.

$MCP\_Mix(MCP\_M)$: *update of* $\widehat{\mathcal{T}}^{ANN}$ *using data from both the environment and the original expert's model.* This version stores in dataset $\dot{\mathcal{D}}$ the triplets $(\dot{s}, \dot{a}, \dot{s}')$

collected from the real-world environment during the last n steps. Then, every n time-steps, it inserts into $\widehat{\mathcal{D}}$ (i.e., the dataset generated by the original expert's model) the triplets in $\dot{\mathcal{D}}$ and removes from $\widehat{\mathcal{D}}$ the triplets that are "close enough" to the triplets just inserted. Finally, MCP_M re-trains $\widehat{\mathcal{T}}^{ANN}$ on the updated $\widehat{\mathcal{D}}$ to adapt the network weights according to both triplets observed from the environment (in parts of the state-action space where they are available) and triplets generated by the original expert's model (in parts of the state-action space where observations from the real-world environment are still not available). After the update $\widehat{\mathcal{T}}^{ANN}$ is used in MCTS simulations for the next n time-steps (i.e., until the next update).

The set of triplets in $\widehat{\mathcal{D}}$ "close enough" to a state-action pair $(\dot{s}, \dot{a}) \in \dot{\mathcal{D}}$ is defined, given the set of thresholds $\mathcal{X} = \{x_0, \ldots, x_v\}$ on state-variables (i.e., a threshold for each state-variable), as

$$\widehat{\mathcal{U}}((\dot{s}, \dot{a}, \dot{s}'), \mathcal{X}) = \{(\widehat{s}, \widehat{a}, \widehat{s}') \in \widehat{\mathcal{D}} | (\dot{a} = \widehat{a}) \wedge \\ \wedge (\dot{s}_0 \in [\widehat{s}_0 \pm x_0]) \wedge \cdots \wedge (\dot{s}_v \in [\widehat{s}_v \pm x_v])\}.$$

Notice that we do not consider $\dot{s}'$ and $\widehat{s}'$ to compute the set of "neighbors" of a triplet since they depend on the specific transition models. Given the dataset $\dot{\mathcal{D}}$ of the last n triplets observed from the real-world environment we therefore introduce them in $\widehat{\mathcal{D}}$ and remove from $\widehat{\mathcal{D}}$ the set $\widehat{\mathcal{U}} = \bigcup_{(\dot{s}, \dot{a}, \dot{s}') \in \dot{\mathcal{D}}} \widehat{\mathcal{U}}((\dot{s}, \dot{a}, \dot{s}'), \mathcal{X})$. $MCP\_Select(MCP\_S)$: *update of* $\widehat{\mathcal{T}}^{ANN}$ *using only data from the environment and usage of the original* $\widehat{\mathcal{T}}^{ANN}$ *for simulated state-action pairs far from observed state-action pairs.* This version works as MCP_R in the first two steps. Namely, it stores in $\dot{\mathcal{D}}$ the triplets $(\dot{s}, \dot{a}, \dot{s}')$ observed in the real-world environment, and every n time-steps performs a re-training of $\widehat{\mathcal{T}}^{ANN}$ using only data in $\dot{\mathcal{D}}$. Afterwards, it uses two different transition models in the MCTS simulations of the subsequent n time-steps. Namely, it uses the expert model $\widehat{\mathcal{T}}^{ANN}$ or the re-trained model $\dot{\mathcal{T}}^{ANN}$, depending on the closeness of the current state-action pair in the simulation $(\tilde{s}, \tilde{a})$ to the state-action pairs of the triplets in $\dot{\mathcal{D}}$. Therefore, given a state-action pair $(\tilde{s}, \tilde{a})$ from a simulation and a set of thresholds $\mathcal{X} = \{x_0, \ldots, x_v\}$ on state-variables, we compute the set of triplets in $\dot{\mathcal{D}}$ close enough to $(\tilde{s}, \tilde{a})$ as

$$\dot{\mathcal{U}}((\tilde{s}, \tilde{a}), \mathcal{X}) = \{(\dot{s}, \dot{a}, \dot{s}') \in \dot{\mathcal{D}} | (\tilde{a} = \dot{a}) \wedge \\ \wedge (\tilde{s}_0 \in [\dot{s}_0 \pm x_0]) \wedge \cdots \wedge (\tilde{s}_v \in [\dot{s}_v \pm x_v])\}.$$

Notice that, also in this case, we do not consider $\dot{s}'$ and $\widehat{s}'$ to compute the set of "neighbors" of a triplet since they depend on the specific transition models. If $\dot{\mathcal{U}}(\tilde{s}, \tilde{a})$ is not empty, we use the network-copy of the expert's model $\dot{\mathcal{T}}^{ANN}$ as a transition model, otherwise we use the network computed from observed triplets $\widehat{\mathcal{T}}^{ANN}$. This selection of the transition model is repeated at each step of the MCTS simulation process, apart in the first n time-steps when only the network-copy of the expert's model $\widehat{\mathcal{T}}^{ANN}$ is available.

5 Application domain

We evaluate the proposed approach on a simulated version of a real-world problem concerning air quality and thermal comfort control in smart buildings (Bianchi et al. 2023; Capuzzo et al. 2022). Let's consider a room (e.g. a meeting room in a company or a classroom in a school/university) provided with sensors measuring carbon dioxide (CO_2) and volatile organic compounds (*VOC*) concentration, indoor and outdoor temperature. Ventilation and air sanification devices are used to manage air renewal and sanitization over time to avoid exceeding threshold limits. In particular, the environment has actuators for opening/closing windows or turning on/off vents and sanitizers. We also assume to have in advance the current daily occupation schedule of the room (i.e. number of persons present in the room at each hour of the day). Some examples of environments with these features are university lecture halls, in which students attend pre-scheduled lectures during the day, or meeting rooms, that must be booked in advance with the number of participants. Taking inspiration from (Teleszewski and Gładyszewska-Fiedoruk 2019; Tang et al. 2016) we developed a simulator of the environment described above to test our framework in silico. We formalized the problem as an MDP whose main elements are listed in the following.

5.1 State space

The *state* contains: (i) the number of persons currently present in the room, (ii) the current concentration of CO_2 in the room, (iii) the current concentration of *VOC* in the room, (iv) the current indoor temperature, (v) the current outdoor temperature.

5.2 Action space

Actions are related to ventilation system, namely *stack ventilation* (open/close windows) or *mechanical ventilation* (turn on vents at low/high speed or turn off), *sanitization* system (turn on/off sanitizers) and combinations of them (see the supplementary material for full details on the action space).

5.3 Transition model

The *transition model* represents the model of the dynamics of the environment that we aim to improve.[1] For each state-action pair the model returns the new state reached by performing the action from the state. The model consists of three *sub-models* used to describe the evolution of CO_2 concentration, *VOC* concentration and indoor temperature, respectively. Each action is characterized in the transition model by a specific value called (ACH) represented with δ_a, which indicates the number of times the entire room air volume is replaced in one hour with that action. This characteristic allows actions to affect the environment air quality (which is part of the reward function

[1] In the following we describe the model we used as a true environment, then in Sect. 6.1 we briefly describe the error that we introduced into the expert's model.

as detailed later) since the air renewal process acts on the concentrations of CO_2 and *VOC*. In the following, we provide an overview of the model. Full equations and the Python code are available in the supplementary material.

Evolution of CO_2 *concentration.* It considers both *stack* and *mechanical ventilation systems.* For stack ventilation we consider the model proposed in (Teleszewski and Gładyszewska-Fiedoruk 2019) that computes the next value of CO_2 concentration i.e., $c_{CO_2}^{i+1}$, as a function of the ACH, the room volume, the number of its occupants, the concentration of CO_2, and the time difference between two time-steps. For mechanical ventilation the value of $c_{CO_2}^{i+1}$ is computed as a function of the ACH, the maximum occupancy of the room and the number of persons in the room. We also consider *active* stack ventilation in unpopulated rooms. In this case $c_{CO_2}^{i+1}$ depends on the concentration of CO_2 and the difference between the indoor and outdoor CO_2 concentrations at time i. Model equations are available in the supplementary material and in the Python code attached.

Evolution of VOC concentration. The estimated concentration of *VOC* depends on the concentration of *VOC* that each person emits in the room, the *VOC* removed by the sanitizer, the number of persons present in the room at time i, the room volume, the *VOC* concentration at time i and the relative variation of CO_2 concentration.

Evolution of internal temperature. The next value for internal temperature T_{in}^{i+1} is computed as a function of the difference between the indoor and the outdoor temperature, the time difference between two subsequent time-steps, the current indoor temperature and the heat generated by people and by sanitizers in operation. The temperature course depends on the value of the difference between the indoor and the outdoor temperature.

Evolution of room occupancy and outdoor temperature. We assume their values over time are given by occupation schedule of the room and weather forecast.

5.4 Reward function

The reward function considers four components: air quality, comfort, energy consumption and the energy factor. The air quality component corresponds to the mean between the reward due to CO_2 concentration and the reward due to *VOC* concentration when there are occupants, otherwise it is set to 1. The values of the rewards due to CO_2 and *VOC* concentrations linearly decrease when the values of c_{CO_2} and c_{VOC} are below their maximum acceptable concentration values. Conversely, these rewards quadratically decrease when their values are below the maximum concentration values we suppose the environment could reach, otherwise they are set to 0. The value of the comfort component is computed as a weighted mean between temperature and noise rewards when there are persons in the room, while it is set to 1 where there are no occupants. The temperature reward is expressed as exponential function of the indoor temperature, while the noise reward and the value of the energy consumption reward are constant values associated to the action performed by the agent. Thus, the noise reward represents a measure of noise pollution. Full details about the reward model are reported in the supplementary material and the code attached.

6 Experiments

We first introduce the experimental setting and then present the results of the empirical analysis performed on the air-quality control domain.

6.1 Experimental setting

We aim to compare the performance of our method with respect to that of baseline model-free and model-based algorithms on the air-quality control domain assuming the availability of an approximated expert's model of the environment. Since tests are performed on a simulated environment, we need a model representing the true environment (not known by the algorithm) and a second model (i.e., a modification of the first model) representing the expert's model.

6.1.1 True environment and expert's model

The transition model described in Sect. 5.3 (and supplementary material) is used as a true environment (i.e., correct transition model). As an approximated model (i.e., expert's model) we instead use inaccurate transition functions adding some errors to the real-world model. In particular, the expert's model used for the *evolution of* CO_2 *concentration* overestimates CO_2 decrease and underestimates CO_2 increase. Contrary to the true environment model for the *evolution of VOC concentration*, the expert's model considers the effect of the action performed in terms of ACH, and it underestimates the concentration of *VOC* produced by each person in the room. Moreover, the expert's model does not consider the relative variation of CO_2 concentration. Finally, in the *evolution of internal temperature* the expert's model does not consider the heat produced by people in the room and by the sanitizers in operation, considered instead by the true model. Full details about the expert's transition models can be found in the supplementary material and in the Python code attached.

6.1.2 Baseline algorithms for performance comparison

In our experiments we compare the three versions of our approach against algorithms that represent the state-of-the-art about MCTS-based planning (two versions), model-free RL and model-based RL. The four baseline algorithms are briefly described in the following.

MCP_O: MCTS planning with the true transition model (oracle). This algorithm uses standard MCTS with UCT and performs simulations with a transition model identical to the true transition model. This planning method is proved to converge to the optimal policy for a large enough number of simulations and it represents a reference that we would like to reach by adapting our inaccurate model to the real model of the environment.

MCP_E: MCTS planning with the expert's transition model (without adaptation). This algorithm uses standard MCTS with UCT and performs simulations with the expert's model, which is different from the true transition model, and it does not

perform any form of adaptation. Hence, this method provides a sub-optimal policy, with actions biased by the simulation error.

PPO: Proximal Policy Optimization (model-free DRL). This algorithm uses the state-of-the-art actor-critic method in which the "actor" learns the policy given the state as input, while the "critic" learns the value of the state, i.e. it receives as input a state and it learns the value of the state given the policy that it is training. Both the "actor" and the "critic" are parametrized as neural networks.

SLBO: Stochastic Lower Bound Optimization (model-based DRL). This algorithm trains a neural network to learn the model of the environment leveraging real-world observations. Then SLBO uses the estimated model to produce samples on which it learns the policy with TRPO.

6.1.3 Implementation details

The generation of the expert's dataset is achieved by running the expert's model with a time interval of 5 min for 800 days (considering different random occupation schedules in different days). We first generate a dataset containing 96,000 state-action-next-state triplets i.e., data for 800 days. Then, this dataset is used to generate the network-copy of the expert's model in our method.

PPO and SLBO do not use the expert's model in their original framework, hence, to make a fair comparison we pre-trained them using the expert's model of the environment as a simulator of the real environment for the same 800 days in the expert's dataset used by our method. Moreover, in SLBO, we used the same structure of neural network used in our method to represent the model of the environment. Since we used an expert's model which has some similarities with the correct one, this process does not negatively impact the performance of PPO and SLBO but it allows them to start from higher performance. We repeated the pre-train process using 5 random seeds and kept the expert's network with the lowest validation loss to be used in MCP_R, MCP_S, and MCP_M, while for PPO and SLBO we chose the model with the highest mean cumulative reward.

After generating the expert's model and pre-training PPO and SLBO with it, we evaluate the performance of the methods and compare them. We consider 10 rooms, each with a specific *100-day-profile* (i.e., occupation schedule and external temperature forecast). For each day, we consider a time interval from 8:00 to 18:00 with 12 samples per hour (i.e., a sampling interval of 5 min). Thus, each day corresponds to $10 \times 12 = 120$ samples, and each 100-day-profile has 12,000 samples for room occupancy, 12,000 samples for external temperature, and an initial value per day for internal temperature. After pre-training, the PPO agent is retrained at the end of each day of execution (i.e., after 120 steps), considering only real-world data acquired during that day, while the SLBO agent is retrained on samples of real-world data collected until the current time instant. Then, both the agents use the new model during the following day of execution (i.e., for 120 steps), namely until the next re-training. It is important to note that PPO, SLBO, and the MCP-based approaches are all trained for a specific number of iterations, not until convergence to avoid overfitting. To quickly simulate yearly executions, we assume each season has a length of 25 days. In different seasons we change the average external temperature. In the testing phase, for MCP_M and

MCP_S, we tested 17 combinations of threshold values (i.e., sets $\mathcal{X}$, as defined in Sect. 4.1.2), and we kept the ones with the highest discounted return. As a stopping criterion, we used a threshold on the number of simulations in MCTS and a threshold for the number of iterations in PPO and SLBO. In particular, we stopped MCTS after 10,000 simulations, PPO after 10 epochs for actor and critic, and SLBO after 75 iterations for dynamics and 50 for TRPO.

6.2 Performance measures

On the 10 rooms described above (100 days per room) we compute the average discounted return ($\overline{\rho_d}$) and the average difference between the discounted returns of different methods ($\overline{\Delta\rho_d}$). Averages are computed across rooms. We define ρ_d as the sum of the rewards collected in all time-steps of episode d. The episode is in general a set of steps between two transition model updates, but in our tests we consider episodes of 1 day since we update the transition model every day. The average discounted return of episode d (i.e., $\overline{\rho_d}$) is then computed as the mean of values ρ_d across all rooms, i.e., $\overline{\rho_d} = (\sum_{y=1}^{z} \rho_{d,y})/z$ where $\rho_{d,y}$ represents discounted return obtained in room y during episode d and z is the total number of rooms. The average difference between the discounted return of two methods (e.g., MCP_R and MCP_E) in episode d is instead computed as $\overline{\Delta\rho_d} = (\sum_{y=1}^{z} \Delta\rho_{d,y})/z$ where z is the total number of rooms, and $\Delta\rho_{d,y}$ represents the difference between the discounted returns of the two methods during episode d in room y, namely, $\Delta\rho_{d,y} = \rho_{d,y}^{M_1} - \rho_{d,y}^{M_2}$, where M_1 and M_2 represent the two methods. The same measure is used to compare any pair of methods.

6.3 Hardware and software

We performed our tests using a Python 3.9 simulator (PyPy and CPython interpreters) on an AMD Ryzen 9 6900HX CPU @ 3.3GHz with 16 GiB of RAM, an Intel Core i7-8750 H CPU @ 2.2 GHz with 16 GiB of RAM and an Intel Core i7-10700 CPU @ 2.90 GHz with 16 GiB of RAM.

6.4 Results

We present our results in a top-down way. First we rank the performance of the three versions of our method to identify the best ones. Then we compare the average performance of the best versions with that of MCP_O, MCP_E, PPO and SLBO. Finally, we provide insight on the functioning of the proposed method and the motivation of the improvement.

6.4.1 Identification of the best versions of our approach

We compute the average difference between the discounted return obtained with each version of our approach and that achieved with MCP_E (averages are computed across the 10 rooms on which the algorithms are tested). The worst performance is achieved

Table 1 Performance comparison between pairs of variants, namely MCP_R, MCP_M and MCP_S

Variant	$\overline{\Delta\rho_d}$	p-value
MCP_R-MCP_M	−0.951	$6.101 * 10^{-5}$
MCP_R-MCP_S	−0.881	$2.341 * 10^{-4}$
MCP_S-MCP_M	−0.070	$7.380 * 10^{-1}$

by MCP_R (i.e., $\overline{\Delta\rho_d} = 12.116$). This variant forgets the expert's knowledge at the first retrain of $\widehat{\mathcal{T}}^{ANN}$ and it requires several days to collect enough observations from the environment to learn a transition model not over-specialized on a small states-action subspace. MCP_S performs better (i.e., $\overline{\Delta\rho_d} = 12.997$). A motivation for this improvement is that it merges the network-copy of the expert's model with the network $\dot{\mathcal{T}}^{ANN}$ trained on observed data, in a more precise way, considering different networks in different state-action subspaces, according to the strategy described in Sect. 4.1.2. *The best performance is achieved by* MCP_M ($\overline{\Delta\rho_d}$ = 13.067). This version is more efficient in merging the information acquired step-by-step from the environment with the knowledge in the expert's model, since it works at a training set level, i.e., it removes samples from the dataset $\widehat{\mathcal{D}}$ as new samples from the real-world environment become available. We performed Student's t-test to verify that these average differences are statistically different from zero obtaining, in all three cases, p-values lower than 0.05.

In Table 1 we show the average difference of discounted return between each pair of versions of our approach (i.e, MCP_R vs MCP_M, MCP_R vs MCP_S and MCP_S vs MCP_M). It emerges that both MCP_M and MCP_S perform better than MCP_R (see the first two lines of the table in which the p-value is less than 0.05), while there is no significant difference between the performance of MCP_M and MCP_S (see the third line of the table in which the p-value is 0.738 hence larger than 0.05). In the following, we fully analyze the functioning and performance of MCP_M and then briefly describe the main differences with MCP_S.

6.4.2 Comparison of average performance: MCP_M vs MCP_E, PPO and SLBO

In this test, we compare the performance of MCP_M against baseline approaches. Figure 2a compares pairs of algorithms in terms of distribution of $\Delta\rho_d$ and $\overline{\Delta\rho_d}$, where distributions and averages are computed across the 10 rooms on which the algorithms are evaluated. The figure confirms that MCP_M outperforms MCTS with the expert's transition model (MCP_E) with an average difference of discounted return $\overline{\Delta\rho_d}$ = 13.067 (as shown in the previous subsection). This is due to the improvement MCP_M introduces into the transition model over time. Figure 2b shows that, on average, MCP_M performs better than PPO ($\overline{\Delta\rho_d}$ = 15.151). This is a reasonable result since PPO is a model-free algorithm used in a non-trivial scenario, thus even though PPO adapts to the environment (and it was pre-trained using the expert's model as a simulated environment), its performance are still worse than the performance of MCP_M, that can leverage the knowledge in the transition model. Finally, Fig. 2c shows that, on average, MCP_M performs better than SLBO ($\overline{\Delta\rho_d}$ = 18.432). This is an important result which shows that the proposed approach can outperform also

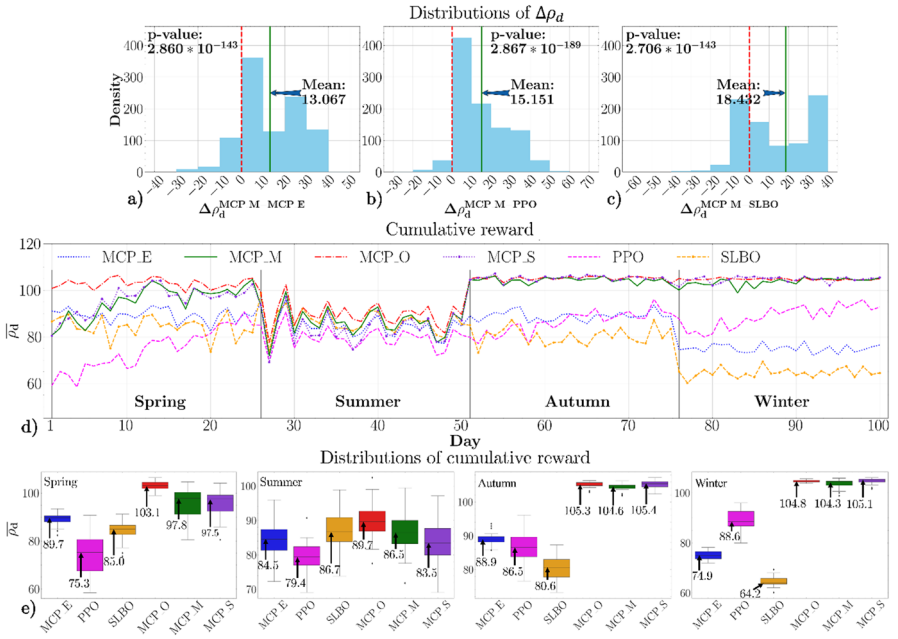

Fig. 2 Density of difference in discounted return between **a** MCP_M and MCP_E, **b** MCP_M and PPO. **c** MCP_M and SLBO. **d** Discounted return over time in 10 rooms obtained by MCP_E, PPO, SLBO, MCP_O, MCP_M, MCP_S. **e** Density of cumulative reward obtained by MCP_E, PPO, SLBO, MCP_O, MCP_M, MCP_S during Spring, Summer, Autumn and Winter

state-of-the-art model-based approaches (notice that the same neural network structure was used in the two methods to represent the transition model).

6.4.3 Comparison of performance over time: MCP_M vs MCP_O, MCP_E, PPO and SLBO

Figure 2d shows the average discounted return $\overline{\rho_d}$ (where the average is computed across the 10 rooms) of MCP_E, MCP_O, MCP_M, PPO, and SLBO, along the 100 days executions on which we performed our tests. The dash-dotted red line, representing the daily discounted return of MCTS with the true transition model (MCP_O), highlights the high performance of the method, which converges to the optimal policy when enough simulations are used. This result is clearly visible in Fig. 2e, that shows the average discounted return $\overline{\rho_d}$ of each approach, grouped by season. In Fig. 2d the solid green line of MCP_M starts below the dotted blue line of MCP_E. This trend reverses for the first time at day 6, confirming the result showed in Fig. 2a. In Fig. 2d we also show that the solid green line of MCP_M manages to reach the dash-dotted red line of MCP_O, stabilizing on the same values from day 50. Figure 2e confirms this result, showing that MCP_M performance increase through seasons until stabilizing on that of MCP_O during autumn (median $\overline{\rho_d}$ of 104.6 and 105.3, respectively) and winter (median $\overline{\rho_d}$ of 104.3 and 104.8, respectively). Moreover, Fig. 2d shows that, at

the beginning, the performance of PPO (dashed pink line) are the worst, but they gradually improve until they outperform MCP_E (dotted blue line) for the first time at day 64. However, PPO never reaches the performance of MCP_O (dash-dotted red line), as MCP_M does, due to a slow performance improvement. This trend is also visible in Fig. 2e, where PPO starts with a median $\overline{\rho_d}$ of 75.3 and it gradually increases until reaching the value of 88.6 in winter. According to Fig. 2d SLBO (dashed orange line with marker o) achieves the worst performance in this test. It outperforms PPO during spring and summer with median $\overline{\rho_d}$ of 85.0 and 86.7 respectively (Fig. 2e), then its performance decreases making SLBO the worst performing approach in autumn and winter achieving median $\overline{\rho_d}$ of 80.6 and 64.2, respectively (Fig. 2e). To the best of our knowledge, we attribute the poor performance of SLBO to the use of TRPO which uses a second-order optimization, which increments the overhead in the training possibly resulting in poor performance (Schulman et al. 2015). These results confirm the average improvement of Fig. 2b, c and provide insight on the origin of this improvement over time. Finally, Fig. 2d shows that, on average, MCP_M, PPO and SLBO usually present a performance decrease as the season changes (vertical black lines) due to the environment change and the need of several days to learn the new environment.

6.4.4 Performance of MCP_S

After analyzing in depth the performance of MCP_M, here we briefly analyze the performance of MCP_S (which are slightly lower than but not statistically different from that of MCP_M). We compared the performance of MCP_S against baseline approaches in terms of distribution of $\overline{\Delta\rho_d}$, computed across the 10 rooms on which the algorithms are evaluated. This test confirms that MCP_S outperforms MCP_E with an average difference of discounted return $\overline{\Delta\rho_d} = 12.997$ due to the improvement MCP_S introduces into the transition model over time. Moreover, MCP_S performs better than PPO ($\overline{\Delta\rho_d} = 15.081$) and SLBO ($\overline{\Delta\rho_d} = 18.362$) showing that our approach can outperform both state-of-the-art model-free and model-based approaches.

In Fig. 2d we show that the performance of MCP_S (dotted purple line with marker o) and MCP_M (solid green line) are very similar in terms of average discounted return along the 100 days executions. Figure 2e confirms this result, highlighting the similarity with MCP_M in each season.

6.4.5 Additional comparison of performance over time: MCP_M vs PPO

In this section, we present the results obtained by comparing the performance of MCP, PPO updated considering only the last day's data, and PPO updated considering all data up to that instant, first over 100 days and then over 500 days.

Performance over 100 days. Focusing on the comparison with PPO, we performed a test by retraining PPO agents each day on all real-world data collected up to that day, over 100 days (PPO_A). Then, we compare its performance with that obtained by retraining agents considering only the data of the last day (PPO_L). As shown in Fig. 3, the performance of PPO trained on all real-world data collected up to that day, PPO_A, is improved only in the first season, but it does not reach the performance of MCP_M anyway. PPO, and other model-free RL methods, require larger amounts of

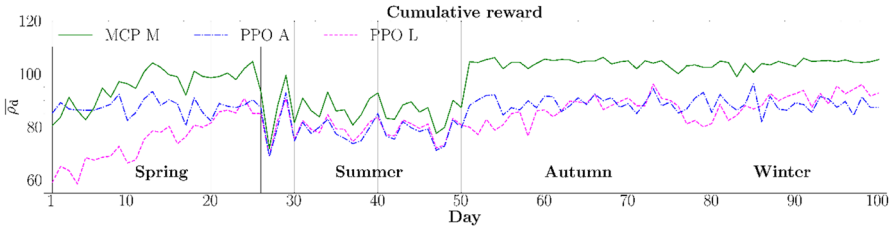

Fig. 3 Discounted return over time in 10 rooms obtained by MCP_M, PPO trained with only the data of the last day (PPO_L), and PPO trained with all data until the current time instant (PPO_A)

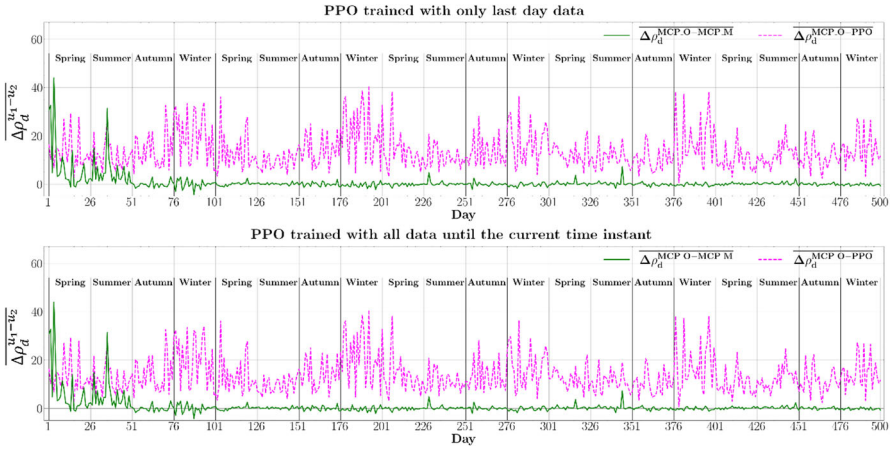

Fig. 4 Difference in discounted return between MCP_O and MCP_M, MCP_O and PPO over a trajectory of 500 days training PPO with **a** only last day data, **b** all data until the current time instant

data to compute an optimal policy. As Fig. 2d shows, PPO's performance improves slower than MCP-based proposed approaches, since they can leverage the explicit representation of the transition model.

Performance over 500 days. PPO manages to reach the same performance as MCP_M after many steps, showing that we used it in a fair way. To confirm this point, Fig. 4a shows the difference in performance between MCP_O and PPO (dashed pink line) and between MCP_O and MCP_M (solid green line) on a trajectory of 500 days. Clearly, both MCP_M and PPO achieve good performance at the end of the trajectory, but MCP_M achieves it earlier than PPO. The theoretical basis of this improvement can also be explained by a parallelism with Dyna-Q, a known model-based RL algorithm. Dyna-Q outperforms Q-learning (a model-free RL algorithm) if an approximated but meaningful transition model is used at the beginning and then updated, because DynaQ can exploit the knowledge in the approximate model. Our method can be seen as an extension of DynaQ in which the model is represented by a neural network and the Q-values are computed by MCTS (which explicitly uses the model and mixes planning and learning) instead of Q-learning. We repeated the same test training PPO with all real-world data collected up to the current time instead of

using only data from the last day, but PPO obtains lower performance and it does not converge to good performance, as we can see in Fig. 4b.

7 Conclusions and future work

The approach presented in this work allows to use MCTS planning also when only an approximated transition model is available. Standard MCTS would generate sub-optimal policies in this case due to the bias introduced by the transition model in the simulations. The proposed method transforms the approximated model in a neural network and improves it as observations about the true dynamics are collected from the environment. Three versions are proposed and compared to state-of-the-art model-free and model-based reinforcement learning techniques on a domain concerning air-quality control. We identified the best version of our method and shown that it outperforms both standard MCTS planning using the approximated transition model and state-of-the-art reinforcement learning methods. Future work mainly concerns the application of the proposed methodology to other domains in order to further evaluate it and possibly extend its capabilities. In particular, the domain on which we tested the approach in this work has continuous state space and discrete action space. We are interested in investigating the applicability to domains with discrete state/action spaces and continuous state/action spaces. Finally, we would like to develop more sophisticated techniques to decide when the retrain of the model is necessary (e.g. change-point detection, conformal prediction) and to test several expert's models to analyze how their accuracy impacts the performance.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11081-024-09896-2.

Acknowledgements The authors acknowledge Luca Marzari for support on the experiments with PPO, and Nicola Renoffio for information exchanges about real-world HVAC domains.

Author Contributions M.Z.: Conceptualization, Methodology, Software, Visualization, Writing - original draft; E.F.: Software, Visualization, Conceptualization; A.C.: Conceptualization, Methodology, Supervision, Writing—review & editing; A.F.: Supervision, Project administration, Writing—review & editing.

Funding Open access funding provided by Università degli Studi di Verona within the CRUI-CARE Agreement. This work was supported by the POR FESR 2014–2020 Work Program of the Veneto Region (Action 1.1.4) through the project No. 10288513 titled "SAFE PLACE. Sistemi IoT per ambienti di vita salubri e sicuri".

Availability of data and materials The authors don't have any research data outside the submitted manuscript file.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethics approval and consent to participate Not applicable.

Consent for publication Not applicable.

References

Bianchi F, Corsi D, Marzari L, Meli D, Trotti F, Zuccotto M, Castellini A, Farinelli A (2023) Safe and efficient reinforcement learning for environmental monitoring. In: Proceedings of Ital-IA 2023: 3rd National Conference on Artificial Intelligence, CEUR-WS.org, CEUR Workshop Proceedings, vol 3486, pp 2610–2615

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games, pp 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

Capuzzo M, Zanella A, Zuccotto M, Cunico F, Cristani M, Castellini A, Farinelli A, Gamberini L (2022) Iot systems for healthy and safe life environments. In: 7th Forum on Research and Technologies for Society and Industry Innovation (RTSI)

Castellini A, Chalkiadakis G, Farinelli A (2019) Influence of state-variable constraints on partially observable Monte Carlo planning. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI, International Joint Conferences on Artificial Intelligence Organization, pp 5540–5546. https://doi.org/10.24963/ijcai.2019/769

Castellini A, Marchesini E, Farinelli A (2021) Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation. Eng Appl Artif Intell 104:104382. https://doi.org/10.1016/j.engappai.2021.104382

Castellini A, Bianchi F, Zorzi E, Simão TD, Farinelli A, Spaan MTJ (2023) Scalable safe policy improvement via Monte Carlo tree search. In: Proceedings of the 40th international conference on machine learning (ICML 2023), PMLR, pp 3732–3756

Chebotar Y, Kalakrishnan M, Yahya A, Li A, Schaal S, Levine S (2017) Path integral guided policy search. In: IEEE international conference on robotics and automation, ICRA. IEEE, pp 3381–3388. https://doi.org/10.1109/ICRA.2017.7989384

Chua K, Calandra R, McAllister R, Levine S (2018) Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In: Proceedings of the 32nd international conference on neural information processing systems, NeurIPS, Curran Associates Inc., pp 4759–4770

Clavera I, Rothfuss J, Schulman J, Fujita Y, Asfour T, Abbeel P (2018) Model-based reinforcement learning via meta-policy optimization. In: Proceedings of the 2nd conference on robot learning, PMLR, proceedings of machine learning research, vol 87, pp 617–629

Deisenroth MP, Rasmussen CE (2011) PILCO: a model-based and data-efficient approach to policy search. In: Proceedings of the 28th international conference on international conference on machine learning, Omnipress, pp 465–472

Deisenroth MP, Fox D, Rasmussen CE (2015) Gaussian processes for data-efficient learning in robotics and control. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/TPAMI.2013.218

Dennunzio A, Formenti E, Manzoni L, Margara L, Porreca AE (2019) On the dynamical behaviour of linear higher-order cellular automata and its decidability. Inf Sci 486:73–87

Dennunzio A, Formenti E, Margara L, Riva S (2023) An algorithmic pipeline for solving equations over discrete dynamical systems modelling hypothesis on real phenomena. J Comput Sci 66:101932. https://doi.org/10.1016/j.jocs.2022.101932

Finn C, Levine S, Abbeel P (2016) Guided cost learning: deep inverse optimal control via policy optimization. In: Proceedings of the 33rd international conference on international conference on machine learning, JMLR.org, vol 48, pp 49–58

Giuliari F, Castellini A, Berra R, Bue AD, Farinelli A, Cristani M, Setti F, Wang Y (2021) Pomp++: Pomcp-based active visual search in unknown indoor environments. In: 2021 IEEE/RSJ interna-

tional conference on intelligent robots and systems (IROS), pp 1523–1530. https://doi.org/10.1109/IROS51168.2021.9635866
Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
Grattarola D, Livi L, Alippi C (2021) Learning graph cellular automata. In: Ranzato M, Beygelzimer A, Dauphin YN, Liang P, Vaughan JW (eds) Proceedings of the 34th international conference on neural information processing systems, NeurIPS. Curran Associates, Inc., pp 20983–20994
Guez A, Silver D, Dayan P (2013) Scalable and efficient bayes-adaptive reinforcement learning based on Monte-Carlo tree search. J Artif Intell Res 48:841–883. https://doi.org/10.1613/jair.4117
Heess N, Wayne G, Silver D, Lillicrap T, Erez T, Tassa Y (2015) Learning continuous control policies by stochastic value gradients. Advances in neural information processing systems. MIT Press, NeurIPS, pp 2944–2952
Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement Learning: a Survey. J Artif Intell Res 4:237–285
Katt S, Oliehoek FA, Amato C (2017) Learning in POMDPs with Monte Carlo tree search. In: Proceedings of the 34th international conference on machine learning - volume 70, JMLR.org, ICML'17, pp 1819–1827
Khansari-Zadeh SM, Billard A (2011) Learning stable nonlinear dynamical systems with gaussian mixture models. IEEE Trans Rob 27(5):943–957. https://doi.org/10.1109/TRO.2011.2159412
Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: Proceedings of the 17th European conference on machine learning. Springer-Verlag, pp 282–293. https://doi.org/10.1007/11871842_29
Kurutach T, Clavera I, Duan Y, Tamar A, Abbeel P (2018) Model-ensemble trust-region policy optimization. In: 6th international conference on learning representations, ICLR, OpenReview.net
Lesort T, Caselles-Dupré H, Ortiz MG, Stoian A, Filliat D (2019) Generative models from the perspective of continual learning. In: IEEE international joint conference on neural networks, IJCNN. IEEE, pp 1–8. https://doi.org/10.1109/IJCNN.2019.8851986
Lesort T, Lomonaco V, Stoian A, Maltoni D, Filliat D, Díaz-Rodríguez N (2020) Continual learning for robotics: definition, framework, learning strategies, opportunities and challenges. Inform Fusion 58:52–68. https://doi.org/10.1016/j.inffus.2019.12.004
Luo F, Xu T, Lai H, Chen X, Zhang W, Yu Y (2022) A survey on model-based reinforcement learning. https://doi.org/10.48550/arXiv.2206.09328
Luo Y, Xu H, Li Y, Tian Y, Darrell T, Ma T (2019) Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In: 7th international conference on learning representations, ICLR, OpenReview.net
Mazzi G, Castellini A, Farinelli A (2021) Rule-based shielding for partially observable Monte-Carlo planning. In: Proceedings of the 31st international conference on automated planning and scheduling, ICAPS. AAAI Press, pp 243–251
Mazzi G, Castellini A, Farinelli A (2023) Risk-aware shielding of partially observable Monte Carlo planning policies. Artif Intell 324:103987
Mazzi G, Meli D, Castellini A, Farinelli A (2023b) Learning logic specifications for soft policy guidance in POMCP. In: Proceedings of the 2023 international conference on autonomous agents and multiagent systems, IFAAMAS, AAMAS '23, pp 373–381
Moerland T, Broekens J, Plaat A, Jonker C (2023) Model-based reinforcement learning: a survey. Found Trends Mach Learn 16(1):1–118. https://doi.org/10.1561/2200000086
Nagabandi A, Kahn G, Fearing RS, Levine S (2018) Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In: IEEE international conference on robotics and automation, ICRA. IEEE Press, pp 7559–7566. https://doi.org/10.1109/ICRA.2018.8463189
Raissi M (2018) Deep hidden physics models: deep learning of nonlinear partial differential equations. J Mach Learn Res 19(1):932–955
Rao AV (2009) A survey of numerical methods for optimal control
Rebuffi SA, Kolesnikov A, Sperl G, Lampert CH (2017) iCaRL: incremental classifier and representation learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, CVPR. IEEE Computer Society, pp 5533–5542. https://doi.org/10.1109/CVPR.2017.587
Richards AG (2005) Robust constrained model predictive control. PhD thesis, Massachusetts Institute of Technology
Ross S, Pineau J, Chaib-draa B, Kreitmann P (2011) A Bayesian approach for learning and planning in partially observable Markov decision processes. J Mach Learn Res 12(48):1729–1770
Russell SJ, Norvig P (2010) Artificial intelligence - a modern approach, 3rd edn. Prentice Hall, London

Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: Proceedings of the 32nd international conference on machine learning. PMLR, vol 37, pp 1889–1897

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. CoRR abs/1707.06347

Silver D, Veness J (2010) Monte-Carlo planning in large POMDPs. In: Advances in neural information processing systems, NeurIPS. Curran Associates Inc., vol 2, pp 2164–2172

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of go with deep neural networks and tree search. Nature 529:484–489. https://doi.org/10.1038/NATURE16961

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge. Nature 550:354–359. https://doi.org/10.1038/NATURE24270

Sutton RS (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Machine learning proceedings. Morgan Kaufmann, pp 216–224. https://doi.org/10.1016/B978-1-55860-141-3.50030-4

Sutton RS (1991) Dyna, an integrated architecture for learning, planning, and reacting. SIGART Bull 2:160–163. https://doi.org/10.1145/122344.122377

Sutton RS, Barto AG (2018) reinforcement learning: an introduction, 2nd edn. A Bradford Book

Tang X, Misztal PK, Nazaroff WW, Goldstein AH (2016) Volatile organic compound emissions from humans indoors. Environmental Science & Technology, pp 12686–12694. https://doi.org/10.1021/acs.est.6b04415

Tassa Y, Erez T, Todorov E (2012) Synthesis and stabilization of complex behaviors through online trajectory optimization. In: IEEE/RSJ international conference on intelligent robots and systems, IROS. IEEE, pp 4906–4913. https://doi.org/10.1109/IROS.2012.6386025

Teleszewski T, Gładyszewska-Fiedoruk K (2019) The concentration of carbon dioxide in conference rooms: a simplified model and experimental verification. Int J Environ Sci Technol 16:8031–8040. https://doi.org/10.1007/s13762-019-02412-5

Wang T, Bao X, Clavera I, Hoang J, Wen Y, Langlois E, Zhang S, Zhang G, Abbeel P, Ba J (2019) Benchmarking model-based reinforcement learning. CoRR abs/1907.02057

Wang Y, Giuliari F, Berra R, Castellini A, Bue AD, Farinelli A, Cristani M, Setti F (2020) POMP: Pomcp-based Online Motion Planning for active visual search in indoor environments. In: 31st British Machine Vision Conference, BMVC. BMVA Press

Zuccotto M, Castellini A, Farinelli A (2022) Learning state-variable relationships for improving POMCP performance. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, SAC. Association for Computing Machinery, pp 739–747

Zuccotto M, Piccinelli M, Marchesini E, Castellini A, Farinelli A (2023) Learning environment properties in Partially Observable Monte Carlo Planning. In: Proceedings of the 8th Italian workshop on artificial intelligence and robotics (AIRO 2022), AI*IA 2022, CEUR-WS.org, CEUR workshop proceedings, vol 3162, pp 50–57