Robust Online Planning with Imperfect Models

Maxim Rostov
Centrum Wiskunde & Informatica,
Dexter Energy B.V.
Amsterdam
maxim.rostan@gmail.com

Michael Kaisers
Centrum Wiskunde & Informatica
Amsterdam
michael.kaisers@cwi.nl

ABSTRACT

Environment models are not always known a priori, and approximating stochastic transition dynamics may introduce errors, especially if only a small amount of data is available and/or model misspecification is a concern. This work introduces a robust decision-time planning method in order to cope with such imprecise models. The objective of *robust planning* is to find a policy with the best guaranteed performance, which we approach by transferring a two-stage minimization-maximization optimization procedure taken from the field of robust control to online planning. We assume a Markov Decision Process underlying the environment and aim for the best worst-case performance within specific model error bounds. To compute solutions, we introduce a family of *locally* robust decision-time planning algorithms, specifically robust Monte Carlo Tree Search (rMCTS). Robust MCTS methods are then evaluated empirically with model error bounded by Wasserstein distance, for which we find the resulting robust policies to yield safer and more uncertainty-aware behavior than their non-robust counterparts. Adaptability in model error bounds and corresponding model minimisers makes robust MCTS extensible for a variety of online planning settings.

KEYWORDS

Planning, Robust Optimization, Reinforcement Learning

1 INTRODUCTION

Engineering and science often solve today's decision-making tasks and challenges by mathematical optimization and computer simulations. Examples of physical or societal systems that are inherently complex and hard to program include estimating a helicopter's flight locomotion [27], and addressing a geographical region's energy production or consumption needs [24] or transportation schedules [15]. Without perfect knowledge about all the components and rules describing these inherently stochastic systems, it is exceedingly difficult to capture their dynamics and create a simulator that is able to accurately account for all relevant real-world factors.

In this work, we address the need to cope with an imperfect simulation model in planning. While learning does not require a description of the simulated system, it may need prohibitively many failures in real systems. Therefore, we here solve the control task with planning techniques, i.e. using the model to derive the optimal control policy [20]. It is not uncommon for traditional planning algorithms to focus on the evaluation with a single model and avoid tackling the problem of robust performance in a real-life setting [11]. This may lead to unstable or sub-optimal behaviour if the model at hand is imprecise, which is often the case when complex real-world systems are digitized [11, 26]. In particular, we assume a model has been learnt from data and/or is known to have errors in its transition probabilities, and focus on its use in *online* planning. While a non-robust planning technique finds a policy that is optimal under one specific model; a robust method aims at a policy that is optimal under the worst case model. Thus, a robust approach needs to resolve the tightly coupled optimisation problem of both maximizing the performance of the control policy and finding the worst-case transition model within a given set of models.

One of the first papers to consider robust control in finite-state, finite-action Markov decision processes (MDP) [25] is on Robust Dynamic Programming [22], where robustness is considered against uncertainty in the transition matrix of an MDP. Consequently, the uncertainty is described by a model space that expresses a set of possible transition functions that are allowed. The authors assume that the set is convex (or it is feasible to find its convex hull) and it obeys the rectangularity property [12]. Under these assumptions they prove that a classical dynamic programming algorithm may be applied to accurately approximate the robust solution. Robust Markov Decision Processes [30] further provide interesting insight into the conditions of rectangularity and types of model space sets while also investigating the performance of robust policies analytically. We build on these theoretical foundations [22, 30] for our methods, and extend planning with the rectangularity assumption by considering alternative model spaces (e.g., Wasserstein distance) and specifically addressing (online) decision-time planning.

Asynchronous dynamic programming [29] has been similarly extended with a number of techniques to integrate planning and robustness. Stochastic Shortest Path problems with uncertain transition dynamics have been tackled with a robust version of the trial-based Real-Time Dynamic Programming (RTDP) algorithm [8]. It is proven that RTDP can be made robust in the common case where the transition probabilities are known to lie in a given interval. In an extension to [8], Delgado et al. [10] propose a method that can act on a general (e.g., nonlinear) set of probability constraints while also having a higher convergence speed due to more efficient sampling of the next state during value updates. Both [8] and [10] use the concepts of locality and relevant states which refer to the fact that only a (small) subspace of the state space is considered during a value update. Therefore, the convergence of the value function is guaranteed (only) when the whole state space is extensively explored, i.e., under an infinite number of state visitations. Unlike our paper, these previous works do not address Monte-Carlo based search methods that have the capacity to efficiently navigate through the state space by using heuristics value functions.

Simão et al. [28] also consider a similar task of finding a safe policy under an estimated, and thus imperfect model. They propose to use the Safe Policy Improvement with Baseline Bootstrapping (SPIBB) algorithm that trains a policy that is guaranteed to perform at least as well as the behavioural policy used to collect the data.

Here, bootstrapping is applied to the trained policy in the state-action pair transitions that were not probed enough in the data set by the behavioural policy. The paper uses robust MDPs to perform safe policy improvements and describes novel (theoretically-based) model space bounds for the worst-case models. While Simão et al. focuses on the offline setting, we apply robust optimization framework in the online setting. Another online algorithm for robust MDP is presented by [19]. In this work, Lim et.al. make the first attempt to perform learning in robust MDPs where it is assumed that some of the state-action pairs are adversarial towards the agent and their parameters can change arbitrarily within some error bounds. In the proposed algorithm, named OLRM, the agent learns the transition parameters of each state-action pair and how to behave robustly in such environments. Here, the authors consider a learning approach which is conceptually different from the planning setting that this paper discusses.

The closest related work introduces a method for robust planning in non-stationary stochastic environments [17]. Just as in our case, the problem is posed as robust planning under uncertain model parameters. The worst-case model parameters are derived by assuming a given model parameter estimate and Lipschitz-continuous evolution of these parameters within an epoch. At each time step, the algorithm performs tree-search on all the state space to find the worst-possible model evolution within Lipschitz constraints.

Our work employs a variation of the Wasserstein optimization method presented in previous work [17] and adopts it as one of the convenient methods to find the worst-case model. While the previous work provides robustness against a non-stationary environment, we establish *local robustness*, i.e. focusing the search of worst-case errors on deviations relevant to the *online* decision-making in the current state. Therefore, the concept of local robustness is similar to *cardinality constrained robustness* [13]. In cardinality constrained robustness, it is assumed that it is unlikely that all the uncertainty parameters change at the same time when analyzing the worst case. Hence, the cardinality of the uncertainty space can be restricted by varying only some parameters, e.g, transition probabilities for the states in the search tree, while the others are modeled with some assumed values.

In this work, we propose a two-stage procedure to tackle the challenge of finding robust and safe policies by online decision-time planning with imperfect models. To this end, we provide a generalization on the assumptions posed by the previous work [13, 17, 22, 30] which helps us to devise a modular and scalable family of robust decision-time planning algorithms based on Monte Carlo Tree Search [9, 16]. The remainder of this paper is structured as follows: Section 2 introduces relevant background, including methods to find non-robust and robust policies. Then, extensions to robust methods are presented in Section 3, including online (tabular) planning methods. Finally, experiments illustrate the empirical value of the proposed methods in Section 4. We conclude with a short summary and directions for future work in Section 5.

2 BACKGROUND

In this section we review the frameworks of model-based reinforcement learning and robust optimization for planning.

2.1 Markov Decision Processes

A Markov Decision Process (MDP) [25] models the decision-making task as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S} \subseteq \mathbb{R}^d$ denotes the state space, $\mathcal{A} \subseteq \mathbb{R}^n$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a state transition probability function that defines the system's dynamics, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function that assesses the agent's performance, and $\gamma \in [0, 1)$ is the discount rate that is applied to the rewards over time. The agent aims to optimize total *return* $G(\tau)$, defined as discounted sum of rewards over state-action trajectory τ.

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t \mathcal{R}\left(s_t, a_t\right). \tag{1}$$

Next, we assume that the dynamics model is defined by the parameters ϕ, and thus we can modify the notation for the MDP model and the transition function to $\mathcal{M}_\phi$ and $\mathcal{P}_\phi$ respectively. Additionally, the agent's policy acting in the environment is a function π parameterized by the vector θ as π_θ. For notational convenience we drop the subscript of π_θ where it aids legibility. We define $\rho_{\phi,\theta}(\tau)$ to denote the trajectory density function that depends on the current policy π_θ, transition function $\mathcal{P}_\phi$ and the stationary initial state distribution μ

$$\rho_{\phi,\theta}(\tau) = \mu\left(s_0\right) \pi_\theta\left(a_0 | s_0\right) \prod_{t=1}^{T-1} \mathcal{P}_\phi\left(s_{t+1} | s_t, a_t\right) \pi_\theta\left(a_t | s_t\right). \tag{2}$$

The focus is on episodic tasks where an episode consists of at most T (discrete) time steps. We can define the expected value of state s conditioned on the current policy π_θ and the transition dynamics parameters ϕ as

$$V^\pi(s) = \mathbb{E}_{\tau \sim \rho_{\phi,\theta}}[G(\tau) \mid s_0 = s] \qquad \forall s \in \mathcal{S}, \tag{3}$$

while the value of state-action pair of s and a is

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \rho_{\phi,\theta}}[G(\tau) \mid s_0 = s,\ a_0 = a] \qquad \forall s \in \mathcal{S};\ \forall a \in \mathcal{A}. \tag{4}$$

The recursive Bellman equation [2] provides the basis of iterative approximation of the value function in subsequent sections.

$$V^\pi(s) = \mathbb{E}_{a \sim \pi_\theta}\left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_\phi\left(s' | s, a\right) V^\pi(s')\right]. \tag{5}$$

2.2 Value Iteration

Value Iteration (VI) is an iterative model-based method that calculates the expected utility under the optimal policy π^* using the values of the neighboring states [29]. More formally, the algorithm performs contraction mapping on the value function space until the value converge to a stable (fixed) point [2]. The recursive update formula looks as following

$$V(s) \leftarrow \max_{a \in \mathcal{A}}\left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_\phi\left(s' | s, a\right) V(s')\right] \quad \forall s \in \mathcal{S}. \tag{6}$$

VI is considered a *background method* [29]. Meaning that it finds the optimal policy/value function for all the states and does not focus on a single (root) state. The method has an advantage of avoiding building an explicit search tree and can be used efficiently with small state spaces. However, it suffers from an obvious pitfall: vanilla VI performs an update on every state in every iteration, thus it can

be computationally intensive and hardly scalable to environments with huge states/action spaces.

2.3 Monte-Carlo Tree Search

An alternative method to background planning is *decision-time planning*. Unlike VI, decision-time techniques focus on finding the optimal action for a single state or a small set of states, representing the specific current (online) decision problem. Frequently, this implies growing a tree of possible path continuations for a root state, i.e., state where the agent resides in the current time step. The direction of the search tree is often determined by a heuristic evaluation function, i.e., *heuristic search* [29], which also helps to determine the order of updates.

An example of a decision-time planning method is Monte-Carlo Tree Search (MCTS) [7, 9]. MCTS is a Monte Carlo rollout algorithm that is adjusted to store value estimates obtained from the past simulations (rollouts) and guide the later simulations towards more promising directions via a heuristic evaluation metric, in our case the UCB tree policy [1]. Hence, MCTS expands and thereby refines the parts of an already built decision tree that have resulted in high evaluations from earlier rollouts.

On each (online) decision step, vanilla MCTS is run from the current state as its root s_{root}, given an MDP model $\mathcal{M}$ and a number of simulations n_{sims} to be performed. Usually, the MDP model is a generative or, as in our work, a distribution model of the environment that is used to produce samples of the rewards and next states. It performs a given number of simulations n_{sims} with a single transition dynamics model $\mathcal{P}$ (included in $\mathcal{M}$). It yields an approximate optimal action for the root state and an estimate of the state-action value function. The best action is executed, and MCTS re-plans in the resulting state. Similar to the existing technique of tree-reuse [23], which may carry forward the sub-tree of the previous search corresponding to the resulting new root state of the next search, we use MCTS as a procedure that accepts an existing tree of estimates, representing state-action value function Q together with the counts of visited states N. In contrast to previous work, we build on a two-stage robust optimisation problem, introduced in the next section, to carry forward estimates over iterations towards a robust solution, as elaborated in Section 3.

2.4 Robust planning solution

Robust optimization aims at extremizing an objective function given that a certain measure of robustness against uncertainty is satisfied [3]. Usually, uncertainty is represented as deterministic variability in the value of the parameters of the problem and/or its solution. Therefore, the most common method of formulating a robust optimization task is the worst-case analysis. This means maximizing the performance of a (policy evaluation) function, f, for the worst-case model within a predefined or learnt set, $\mathcal{U}$, of (MDP) models.

In the context of planning and reinforcement learning, robust optimization is used to produce a policy that provides the best performance under the worst-case environment model. Consequently, a solution to a reinforcement learning task involves optimizing for an objective that encourages lower regret under the worst-case transition dynamics model in a model space set. Following notations from the related work [14, 22] the robust policy evaluation function can be written as

$$f_{\mathcal{U}}(\theta) = \min_{\phi \in \mathcal{U}_\phi} \mathbb{E}_{\tau \sim \rho_{\phi,\theta}} [G(\tau)] \tag{7}$$

where $\mathcal{U}_\phi$ is the set of all possible model space parameter values. Function $f_{\mathcal{U}}(\cdot)$ is also called *worst case expected return* (WCER) [30]. When applied to Equation 3, the worst case value function arises as

$$V^{\pi}_{\phi_{min}}(s) = \min_{\phi \in \mathcal{U}_\phi} \mathbb{E}_{\tau \sim \rho_{\phi,\theta}} [G(\tau) \mid s_0 = s] \qquad \forall s \in \mathcal{S}, \tag{8}$$

which is equivalent to Equation 2 in [30] where authors tackle a similar task.

There are benefits of using such formulation of the value function. For example, in the absence of a priori information about the transitions that are considered non-robust or uncertain, such as a prior in the Bayesian approach [4], we are still able to derive control policies with guaranteed minimal performance, measured by WCER. Additionally, choosing the model space set $\mathcal{U}$ adjusts the robustness level that is desired for an algorithm. If the error bound (or uncertainty level) is set to zero, the model space shrinks and we arrive at a singleton $\mathcal{U}$ that corresponds to working with classical RL methods, where the robust value function Equation 8 is equivalent to the plain MDP formulation of the value function, Equation 3. Increasing the error bound and considering larger model spaces corresponds to encouraging a higher robustness level, as a wider range of possible transition models is being considered.

Maximization of worst case expected return is a well researched problem which has intertwined robust optimization, dynamic programming and reinforcement learning [3, 14, 22]. The main condition that has been exploited to solve this problem in finite time is the rectangularity assumption, introduced from the sub field of utility theory in economics [12]. The idea behind rectangularity of a model space is that transition probabilities are given by independent marginals. The problem can then be understood in game theoretic terms as a zero-sum game between the agent, choosing a (robust) policy, and adversarial nature, choosing the worst-case model within a constrained model space [14].

Consequently, the worst case expected return can be found by solving a two-stage minimax problem. First, a policy π is used to estimate the worst-case MDP transition model. Second, the worst-case model is employed to derive the value function. Iteration may be needed to converge to the robust policy $\tilde{\pi}$.

$$Q^{\tilde{\pi}}(s,a) = \min_{\phi \in \mathcal{U}_\phi} \left[\mathcal{R}(s,a) + \gamma V^{\pi}_{\phi}(s') \right] \qquad \forall s, s' \in \mathcal{S};\ \forall a \in \mathcal{A}, \tag{9}$$

$$V^{\tilde{\pi}}(s) = \max_{a \in \mathcal{A}} Q^{\tilde{\pi}}(s,a) \qquad \forall s \in \mathcal{S};\ \forall a \in \mathcal{A} \tag{10}$$

2.5 Robust Value Iteration

Iyengar [14] introduced a variant of Value Iteration (Section 2.2) to address robust planning. The update formula for robust VI (rVI) incorporates both maximization and minimization stages and accurately approximates the solution of robust Markov Decision Processes [22, 30].

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left[\min_{\phi \in \mathcal{U}_\phi} \left(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_\phi \left(s' | s, a\right) V(s') \right) \right], \quad \forall s \in \mathcal{S} \tag{11}$$

The value function from Equation 11 sequentially solves minimization and maximization problems. Where under certain conditions (e.g., rectangularity [12]) the solution of the minimisation stage can be found with high computational efficiency [14, 22]. This technique serves as the benchmark *offline* reference comparison for our new online planning algorithms.

3 ROBUST DECISION-TIME PLANNING

Having introduced a general methodology of solving robust planning with the two-stage minimax procedure, we start this section by describing two types of operators that can be use to solve the minimization step. Subsequently, we introduce the concept of local robustness, and present our novel robust Monte-Carlo Tree Search methods.

3.1 Model spaces and worst case models

Robust planning requires a procedure to estimate the worst-case MDP transition dynamics model $\mathcal{P}_{min} = \mathcal{P}_{\phi_{min}}$. To build towards robust decision-time planning, we here first transfer a solution from the literature on value iteration to the minimisation stage in decision-time planning (named *direct projection*), and then introduce an approximation (named *indirect projection*) that yields less computational overhead when iteratively approaching a robust solution with an MCTS variant.

Conventionally, $\mathcal{P}_{min}$ comes from a model space $\mathcal{U}$ constructed around the current estimate of the transition dynamics model $\mathcal{P}_0$, which represents the imperfect model available to the planner. In this work we focus on rectangular model space sets. We model the uncertainty in the transition dynamics such that the transition probability $p_0(\cdot|s,a)$ for each state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ is selected from a set $\mathcal{U}(p_0(\cdot|s,a))$, which is derived independently of the transition probabilities of other state-action pairs. Adhering to previous naming conventions [22], we define the process of finding a candidate worst-case model from the reference dynamics model as a *projection* operator. Additionally, we categorize the projection operators into two groups by their ability to move directly or indirectly to the border of the model space, i.e., where the worst-case model is situated.

We can describe the problem of finding the worst-case model from a rectangular model space as the situation when the environment shifts some amount of the probability mass on to the transition states with the lowest state-action value. In this paper, we focus on finite state/action space MDP's and, for notational simplicity, sometimes referring to the parameter vector values of a discrete transition function $p(\cdot|s,a)$ as $p^{s,a}$. The resulting (linear) minimization program is

$$\begin{aligned} \underset{p(\cdot|s,a)}{\text{minimize}} \quad & \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s'), \\ \text{subject to} \quad & \sum_{s' \in \mathcal{S}} p(s'|s,a) = 1, \\ & p(s'|s,a) \in [0,1], \qquad \forall s' \in \mathcal{S} \\ & d\left(p(\cdot|s,a), p_0(\cdot|s,a)\right) \leq \epsilon \end{aligned} \tag{12}$$

where d is a distance metric applied to the parameter space of $p(\cdot|s,a)$. Formulation 12 encodes the constraints on properties of a transition probability function (i.e., positive and sum to one) together with the constraint of $\mathcal{U}(p_0(\cdot|s,a),\epsilon)$ which guarantees that the distance between a candidate worst-case $p(\cdot|s,a)$ and the reference transition $p_0(\cdot|s,a)$ cannot be greater than ϵ. Edges of a geometry (convex polytope) represented by these constraints provide a frontier where one of the points yields the lowest $Q(s,a)$ for a given policy π.

Constraints described in 12, as well as the choice of the distance metric and epsilon value are crucial to creating $\mathcal{U}(p_0(\cdot|s,a),\epsilon)$ that encourages the desired robustness properties. If the distance metric d is convex, the problem 12 becomes a convex optimization and it can be solved analytically. Therefore, we can efficiently apply a *direct projection* operator or Algorithm 1 to perform minimization.

Algorithm 1: Direct Projection

Input : $\mathcal{P}_0$, reference transition model
V^{π}, estimated value function
$S \times A$, state/action (sub)-space for the projection
ϵ, error bound
Output: $\mathcal{P}_{min}$, worst-case transition model

```
for s, a in S × A do
    P_min(·|s,a) ← solving (12) for U(p_0(·|s,a), ε)
end
```

If the program described in 12 cannot be solved efficiently on each iteration step, e.g., due to a chosen distance metric being non-convex, we are still able to approximate the solution to the minimization stage using numerical techniques. Here, the projection operator can be viewed as a gradient step towards the boundary of $\mathcal{U}$. The direction of this step is then identified by the point ∇_p where the transition probability mass is shifted to the lowest value (next) state s'. Hence, the point vector (minimization point) ∇_p consists of all zeros except one entry with the value of 1, i.e., lowest value transition. If there is more than one state yielding the lowest value, a heuristic $h(\cdot)$ is employed to break ties, e.g., 1 is assigned to the closest state to the (s,a)-pair in question. Next, we can introduce step sizes Λ which are used to descend (i.e., minimize). A numerical method such as *dichotomous search* [21] allows us to move towards the boundary of $\mathcal{U}(p_0(\cdot|s,a),\epsilon)$. Compiling all the steps, Algorithm 2 shows a simple procedure able to converge to a (local) minimum of LP 12 by moving toward the model space boundary and iteratively decreasing the gradient step size (with the rate κ). If the model changes during planning, for example due to a new reference dynamics $\mathcal{P}_0$, one can reset the step size values for all state-action pairs (Λ) in order to begin a new minimization stage.

3.2 Local robustness

A problem arises when the input to the robust optimization problem 12 is not defined over the whole state space $\mathcal{S}$. There are often cases of imperfect models when $V^{\pi}(s')$ is not known for all s', i.e., only an estimate $\hat{V}^{\pi}(s')$ on the state sub-space $\tilde{\mathcal{S}} \subset \mathcal{S}$ is available. We can then relax the assumption of having the minimization step for all (s,a)'s pairs and adjust the objective function of Program 12 accordingly by replacing the value function V^{π} with its estimate $\hat{V}^{\pi}$ and defining both the transition function $p(\cdot|s,a)$ and value function

Algorithm 2: Indirect Projection

Input : $\mathcal{P}_0$, reference transition model
$\hat{\mathcal{P}}_{min}$, previous estimate of the worst-case model
V^{π}, estimated value function
$\mathcal{S} \times \mathcal{A}$, state/action (sub)-space for the projection
ϵ, error bound
Λ, initial step sizes
κ, step size decrease rate (default 0.5)

Output : $\hat{\mathcal{P}}_{min}$, Λ, worst-case transition model estimate & step size values

1 $\hat{\mathcal{P}}_{min} \xleftarrow{copy} \mathcal{P}_0$
2 **for** s, a **in** $\mathcal{S} \times \mathcal{A}$ **do**
3 $\quad \nabla_p \leftarrow h(s)$
4 $\quad \hat{p}_{c,min}(\cdot|s,a) \leftarrow \Lambda[s,a] \cdot \nabla_p + (1 - \Lambda[s,a]) \cdot \hat{\mathcal{P}}_{min}(\cdot|s,a)$
5 $\quad$ **if** $d(\hat{p}_{c,min}(\cdot|s,a), \hat{p}_0(\cdot|s,a)) > \epsilon$ **then**
6 $\quad\quad \Lambda[s,a] \leftarrow \Lambda[s,a] \cdot \kappa$
7 $\quad$ **else**
8 $\quad\quad \hat{\mathcal{P}}_{min}(\cdot|s,a) \leftarrow \hat{p}_{c,min}(\cdot|s,a)$
9 $\quad$ **end**
10 **end**

Algorithm 3: rMCTS-Batched

Input : $\mathcal{M}_0$, reference MDP $\mathcal{M}_0 = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_0, \mathcal{R}, \gamma \rangle$
s_{root}, root state
$n_{batches}$, number of batches
n_{sims}, number of simulations for a batch
ϵ, error bound

Output : a, robust action $\arg\max_a Q(s_{root}, a)$

Init: $\mathrm{Q_r}$, $\mathrm{N_r}$, empty hash tables

1 $\hat{\mathcal{P}}_{min} \xleftarrow{copy} \mathcal{P}_0$
2 **for** i **in** $[1, ..., n_{batches}]$ **do**
3 $\quad \mathcal{M}_{min} = \langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}_{min}, \mathcal{R}, \gamma \rangle$
4 $\quad \mathrm{Q_r}, \mathrm{N_r} \leftarrow \mathrm{MCTS}(\mathrm{s_{root}}, \mathcal{M}_{min}, \mathrm{n_{sims}}, \mathrm{Q_r}, \mathrm{N_r})$
5 $\quad \mathrm{V_r} \leftarrow \mathrm{TakeArgMax}(\mathrm{Q_r})$
6 $\quad \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \leftarrow \mathrm{GetStateActionTuples}(\mathrm{Q_r})$
7 $\quad \hat{\mathcal{P}}_{min} \leftarrow$ Algorithm 1$\left(\mathcal{P}_0, \mathrm{V_r}, \tilde{\mathcal{S}} \times \tilde{\mathcal{A}}, \epsilon\right)$
8 **end**

$\hat{V}^{\pi}(s')$ over $\tilde{\mathcal{S}}$ instead of $\mathcal{S}$. We call this case of robustness *local robustness*. Hence, a locally robust policy is derived by making above mentioned adjustments to the program 12 and proceeding with the same robust optimization methodology. One example where local robust solution could be useful is when approximating the DP solution via a decision-time planning algorithm, e.g., Monte Carlo Tree Search [9]. MCTS only builds a sub-tree of the whole MDP space, hence the robustness can only be obtained with respect to the explored subspace $\tilde{\mathcal{S}} \subset \mathcal{S}$.

3.3 Robust MCTS-batched

Consider an algorithm that is a simple combination of the direct projection operator, Algorithm 1, and the MCTS procedure. A plain MCTS algorithm performs N_{sims} in a set amount of times. In a robust version of MCTS, Algorithm 3, we divide N_{sims} into batches, and update the estimation of the worst-case dynamics $\mathcal{P}_{min}$ after each batch. In each iteration, the updated $\mathcal{P}_{min}$ is used in the MCTS algorithm and the improved worst case value function is calculated.

The implementation presented in Algorithm 3 could be seen as an iterative adjustment of the control policy to be more robust with respect to an adversarial environment. Since the robust value function is being used in the projection operator (line 7), each new estimate of the worst-case transition function $\hat{\mathcal{P}}_{min}$ is the nature's best response to the (previous batch's) robust policy. Imagine that the nature's best response to the non-robust optimal control policy π^* derived from interactions with the initial sample model $\mathcal{P}_0$ is $\mathcal{P}_1^{min}$, i.e., $\mathcal{P}_1^{min}$ is $BR(\pi^*(\mathcal{P}_0))$. Next, the agent adjusts its policy to perform optimally (robustly) under $\mathcal{P}_1^{min}$, and the adversary environment then changes to $\mathcal{P}_2^{min}$, i.e, $\mathcal{P}_2^{min}$ is $BR\left(\pi^*(\mathcal{P}_1^{min})\right)$. The game continues until neither the agent nor nature has the incentive to change its best response strategy. Note that due to the averaging behaviour of MCTS, values are smoothed out even if the worst case model varies between batches, which stabilizes the robust value estimates.

3.4 Robust MCTS-iterative

In order to maximise the information flow between the minimisation and maximisation stage, one may want to execute the previously introduced batched algorithm with a batch size of one. However, the repeated execution of a minimization program produce computational overhead for small batches. The algorithm presented next is an iterative version of the method from the previous section with a batch size of one (a single iteration), but using the indirect projection method instead (Section 3.1).

Unlike Algorithm 3, Algorithm 4 always estimates both robust and non-robust value functions (line 6 and 11). The non-robust state values are used with Algorithm 2 to update the current estimate of the worst-case transition function $\hat{\mathcal{P}}_{min}$. Robust state-action values are later derived with respect to $\hat{\mathcal{P}}_{min}$ so that after each simulation the current estimation of the robust policy is presented. However, because the indirect projection is performed with non-robust values, the resulting robust policy is the agent's best response to only $\mathcal{P}_1^{min}$ which is in turn nature's $BR(\pi^*(\mathcal{P}_0))$. Algorithm 4, in this simple form, does not allow for a game-like robust policy as in the batched version. However, it can be expected to converge sooner because the robust policy is being adjusted in parallel to estimation of the non-robust policy. We can easily change Algorithm 2 on line 8 to take in the robust estimate of value function $\mathrm{V_r}$. In this case however, we need to make sure that the learning rates Λ are reset periodically, because that will allow the indirect projection operator to move to a new $\mathcal{P}_i^{min}$ after each reset. Notice, an optional warm-up period on line 2. This step is added in order to explore the state space before making the first update to $\mathcal{P}_{min}$, thus increasing the accuracy of the first few updates. This enable us to increase the step size Λ, thus increasing the convergence speed.

Algorithm 4: rMCTS-Iterative

Input : $\mathcal{M}_0$, reference MDP $\mathcal{M}_0 = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_0, \mathcal{R}, \gamma \rangle$
s_{root}, root state
n_{sims}, number of iterations
n_{warmup}, number of warm-up iterations
Λ, step size
ϵ, error bound

Output : a, robust action $\arg\max_a Q(s_{root}, a)$

Init: Q, N, $\mathrm{Q_r}$, $\mathrm{N_r}$, empty hash tables

$\hat{\mathcal{P}}_{min} \xleftarrow{copy} \mathcal{P}_0$
Q, N $\leftarrow$ MCTS(s_{root}, $\mathcal{M}$, n_{warmup}, Q, N)
V $\leftarrow$ TakeArgMax(Q)
for i **in** $[1, ..., n_{sims}]$ **do**
  Q, N $\leftarrow$ MCTS(s_{root}, $\mathcal{M}$, 1, Q, N)
  V $\leftarrow$ TakeArgMax(Q)
  $\tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \leftarrow$ GetStateActionTuples(Q)
  $\hat{\mathcal{P}}_{min}, \Lambda \leftarrow$ Algorithm 2$\left(\mathcal{P}_0, \hat{\mathcal{P}}_{min}, \mathrm{V}, \tilde{\mathcal{S}} \times \tilde{\mathcal{A}}, \epsilon, \Lambda\right)$
  $\mathcal{M}_{min} = \langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}_{min}, \mathcal{R}, \gamma \rangle$
  $\mathrm{Q_r}$, $\mathrm{N_r} \leftarrow$ MCTS(s_{root}, $\mathcal{M}_{min}$, 1, $\mathrm{Q_r}$, $\mathrm{N_r}$)
  $\mathrm{V_r} \leftarrow$ TakeArgMax($\mathrm{Q_r}$)
end

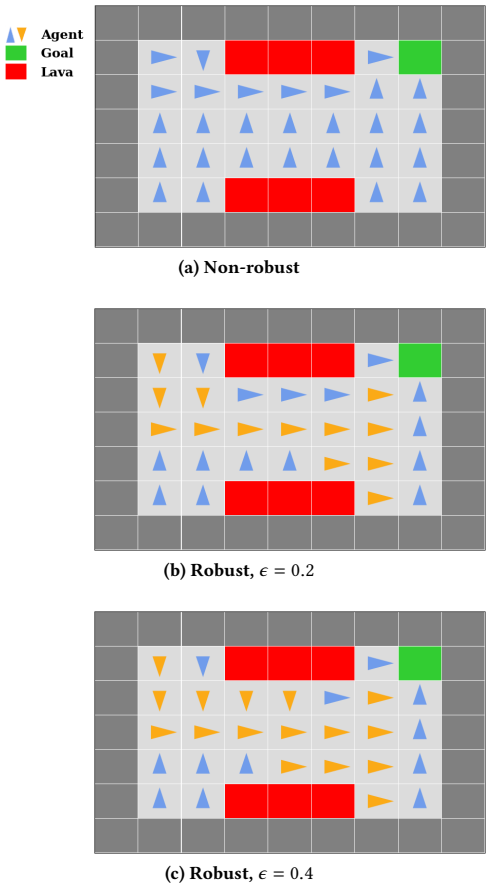
4 EXPERIMENTS AND RESULTS

This section describes our experimental results which we derive by testing our algorithms in two types of grid-world environments from the literature. First, we consider a scenario that highlights the importance of robust performance under a deterministic transition model. Then, we consider a similar setting but with a stochastic environment. The performances of robust and non-robust policies are evaluated quantitatively and qualitatively. Last, we show that in these environments decision-time robust versions of MCTS (rMCTS) are effective in estimating the robust value function and converge to the same solution as robust Value Iteration (Section 2.5).

4.1 Deterministic environment: Lava World

We first test our methods on an environment from DeepMind's AI Safety Gridworlds suite [18] whose original purpose was to assess the ability of a RL agent to (safely) generalise well between train and test settings, i.e., addressing the reality gap [5, 18]. A solution to this problem consists of finding a policy from an initial state that guides the agent safely to the goal state (lime green) without falling into the lava (red cells). The agent is represented with a triangle, the pointing direction of each triangle indicates an action that the agent's policy dictates it to take in that state. With our experiments we show how non-robust and robust methods solve the training environment and examine the performance of both types of methods under perturbations to the transition model dynamics.

Figure 1 displays the differences in the behaviour of robust and non-robust policies. Figure 1 (a) shows the policy derived from running non-robust Value Iteration and MCTS algorithms. We can see that the policy often leads to the states adjacent to the lava at the top of the figure. Our robust planning methods rVI, rMCTS-Batched and rMCTS-Iterative use Wasserstein distance limited errors in the transition function to change this behaviour. Figure 1 (b) and (c) indicate the policies that the robust methods converge to for $\epsilon = 0.2$ and 0.4, respectively. Note that due to the relatively small size of the environment, the online planning algorithms (rMCTS variants) are able to find the same policy and value function as their offline counterpart, robust Value Iteration. It is seen from the Figures 1 that if the agent starts from either of the states in the first two columns, it will never pass near lava. Notice that if the agent starts in some of the states adjacent to lava, it may still choose to pass next to it. This is due to those states being in-between the top and bottom lava, thus the eventual (safety) benefit of moving away from lava is overcome by moving to the state closer to the goal.



(a) Non-robust

(b) Robust, $\epsilon = 0.2$

(c) Robust, $\epsilon = 0.4$

Figure 1: (a) Policy derived for Lava World environment [18] by the non-robust methods (VI and MCTS); (b) policy derived with robust methods (rVI, rMCTS-batched and rMCTS-iterative) given the Wasserstein distance of $\epsilon = 0.2$ and (c) Wasserstein distance of $\epsilon = 0.4$. Orange triangles represent actions that differ between robust and non-robust solutions. We can see that the robust solutions try to avoid transitions that are spatially close to the worst-case states (red cells), thus considering properties of the state space when deriving the policy.

Table 1: Average discounted return over 10000 runs (standard deviation in brackets). Policy performances under different transition functions is considered where the worst-case variation of $\mathcal{P}$ (e.g., $\mathcal{P}^{rMCTSi}_{min}$) is taken after a robust method (e.g., rMCTS-Iterative) has approximately converged in value for the root state, i.e., changes between consequent value updates are less than a threshold $\zeta = 0.001$. Discounting is applied at the rate 0.95 and the robust polices are derived with $\epsilon = 0.2$ Wasserstein distance.

(a) Lava World (Distribution shift) environment from [18]

	$\mathcal{P}_0$	$\mathcal{P}^{rVI}_{min}$	$\mathcal{P}^{rMCTSb}_{min}$	$\mathcal{P}^{rMCTSi}_{min}$	$\mathcal{P}_{test}$	$\mathcal{P}_0 + \xi$
π^{MCTS}	**0.698**	0.212	0.223	0.209	0.340	0.575 (0.239)
π^{rVI}	0.630	**0.309**	**0.302**	**0.304**	**0.625**	0.573 (0.123)
π^{rMCTSb}	0.626	**0.308**	**0.307**	**0.308**	**0.638**	0.574 (0.123)
π^{rMCTSi}	0.626	**0.302**	**0.308**	**0.302**	**0.631**	0.575 (0.120)

(b) Variation of the Frozen Lake environment from [6]

$\mathcal{P}_0$	$\mathcal{P}^{rVI}_{min}$	$\mathcal{P}^{rMCTSb}_{min}$	$\mathcal{P}^{rMCTSi}_{min}$	$\mathcal{P}_0 + \xi$
0.511 (0.198)	0.284	0.283	0.2855	0.481 (0.201)
0.504 (0.173)	**0.303**	**0.309**	**0.304**	0.477 (0.18)
0.501 (0.172)	**0.309**	**0.302**	**0.309**	0.478 (0.179)
0.503 (0.171)	**0.307**	**0.314**	**0.309**	0.475 (0.179)

Table 1 (a) shows the performance of robust and non-robust algorithms as measured by the expected discounted return (Equation 1). The evaluation is performed by generating trajectories from a single root state (top left cell) while the mean discounted return over 10000 runs/paths is recorded. Boldfaced numbers represent significantly greater differences, judged by the paired Wilcox test [31], in the returns of policies derived by robust and non-robust methods. As expected, non-robust methods VI & MCTS, outperform the robust methods rVI, rMCTS-Batched & r-MCTS-Iterative (all with the $\epsilon = 0.2$ Wasserstein distance) when the sample transition model $\mathcal{P}_0$ is indeed true and presented at test time. Unsurprisingly, the robust methods are significantly better under the worst-case model. Since the online robust methods are able to explore the whole state space extensively, they arrive to the same worst-case model projection as robust Value Iteration. In other words, $\mathcal{P}^{rMCTSb}_{min} = \mathcal{P}^{rMCTSi}_{min} = \mathcal{P}^{rVI}_{min}$ where $\mathcal{P}_{min}$ (see Algorithms 3, 4) is a worst-case model estimate $\hat{\mathcal{P}}_{min}$ from a robust method, after it has approximately converged in value for the root state, i.e., changes between consequent value updates are less than a threshold $\zeta = 0.001$. This explains the identical performance between the columns $\mathcal{P}^{rMCTSb}_{min}$, $\mathcal{P}^{rMCTSi}_{min}$, $\mathcal{P}^{rVI}_{min}$, i.e, differences are random fluctuations. It is clear that the robust algorithms are trained to perform better under the worst-case model. Hence, we conduct two additional tests (last two columns) that serve as proxies for real-world deployment. Column $\mathcal{P}_{test}$ holds discounted returns under the original problem posed by AI Safety Gridworlds [18]. Here, the test environment differs from the train environment presented in Figure 1. At test time, either top or bottom lava cells (equally likely) are shifted three rows down or up, i.e., creating 2x3 lava pool in the mid-bottom/mid-top of the grid. Robust algorithms are able to solve this environment change and deliver superior results compared to non-robust methods. We also introduce a column $\mathcal{P}_0 + \xi$. Here, we assume that the true transition function presented at test time is close to the reference model $\mathcal{P}_0$ but the dynamics are also exposed to random perturbations, ξ. Therefore, we mix the reference transition probability function $\mathcal{P}_0$ with a probability matrix ξ in the following manner: for each $s, a \in \mathcal{S} \times \mathcal{A}$, we take $(1-z) \cdot \mathcal{P}_0(\cdot|s,a) + z \cdot \xi(\cdot|s,a)$ where $z \sim Uniform(0, \epsilon)$ and $\xi(\cdot|s)$ is a probability vector where all the mass is equally redistributed around four closest (as measured by L1 distance) to s states. In this manner, we assume that the true transition function can diverge from the estimate $\mathcal{P}_0$ within the geometrical bounds of the state space. This evaluation shows that the discounted return of both robust and non-robust policies is comparable on this proxy, while robust methods have a smaller standard deviation (in brackets) and fewer extreme outcomes.

4.2 Stochastic environment: Frozen Lake

In this section, we examine the behaviour of robust and non-robust policies using a variation of the well-known *FrozenLake* problem from OpenAI gym [6]. In our example, the agent needs to reach one of the two goal positions in a 3x8 ice world while avoiding falling into the water. This environment is particularly challenging due to the high-level of stochasticity in transition dynamics: the agent only moves towards the intended direction one-third of the time, and into one of the two tangential directions the rest. On the Figure 2, one can see a visual of the studied environment. The lime green cells are the goal states and transitions to these states yield the reward of 1. All other transitions yield the reward of 0, including the ones that lead to the red cell that represent a 'hole', i.e., a terminal state where the agent fails the episode.

We investigate the consequences of using robust algorithms in such stochastic environment and record the resulting changes in the agent's behaviour and performance. Figure 2 shows the differences in behaviour between the non-robust and robust solutions under the discount rate of 0.95. Figure 2 (a) represents the non-robust policy as found by Value Iteration and plain MCTS algorithms. Figure 2 (b) depicts the robust policy as indicated by robust Value Iteration and rMCTS versions with Wasserstein distance set to $\epsilon = 0.2$. We can see that the robust policy diverges from the non-robust one for states with an orange triangle. Robust planning tries to avoid going to the goal state in the right-bottom corner as it is situated close to the terminal state. This represents the reluctance of robust planning solutions to move towards the state sub-space that is (spatially) near the risky negative terminal state.

Table 1 (b) shows the performance of robust and non-robust algorithms quantitatively. We generate trajectories at a single root state, on Figure 2 (b) the cell with the most left orange triangle. We report the mean discounted return over 10000 runs/trajectories. As seen from the $\mathcal{P}_0$ column, the robust policy yields slightly worse performance than the non-robust one, however it is able to reduce the standard deviation of the returns (shown in brackets). As in the deterministic environment, the performance of robust methods is

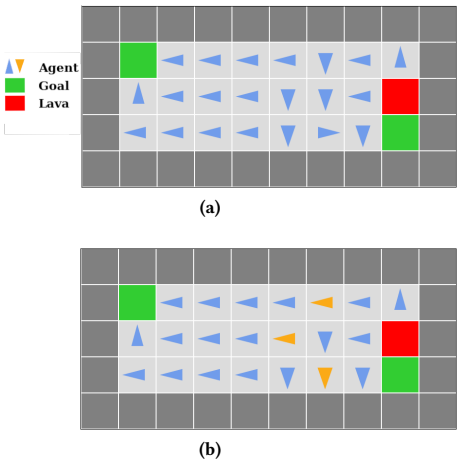


Figure 2: Our version of the Frozen Lake environment from [6]. The agent starts at a random position and needs to reach a goal state (green) while avoiding the lava state (red). (a) Non-robust policy derived from VI and MCTS (b) Robust policy derived with the Wasserstein distance and $\epsilon = 0.2$. Orange triangles represent actions that differ between robust and non-robust solutions.

significantly better under the worst-case model. Lastly, on our real-world transition dynamics's proxy (transition model parameterized by $\mathcal{P}_0 + \xi$) the robust algorithms again appear to perform on par with their non-robust counterparts while having lower standard deviation in the returns.

The experiments on Lava World and Frozen Lake environments indicate that if the given planning model was indeed accurately estimated, the average performance of robust methods can be inferior to that of non-robust methods. However, if the real-life dynamics bear some model error, planning with our robust algorithms is likely to yield more stable performance compared to the non-robust techniques.

5 DISCUSSION & CONCLUSION

In this paper, we extended the existing research on online decision-time planning with imperfect models, i.e., models that are assumed to have errors in their transition dynamics. To that end, we employ the robust optimization framework where the worst-case analysis of the value function is formalized as a two-stage minimization-maximization process. For the minimization stage, we define two types of projection operators, indirect and direct projections, that are modular and can be easily inserted into planning algorithms to make them robust. Next, we introduce the novel concept of local robustness that allows us to incorporate robustness in decision-time planning. Specifically, we devise a family of robust Monte-Carlo Tree Search algorithms that are able to utilize an imperfect model of the environment to perform robust planning. Finally, our experiments on grid-wold domains provide evidence of the efficacy of the proposed approaches, which suggest to be advantageous under decision making with transition model misspecifications.

While our preliminary evaluation has exposed the potential of robust MCTS solutions, experiments in larger (state space) environments are needed to fully expose the benefits of robust decision-time algorithms. In particular, MDPs with a huge state space and many irrelevant states (i.e., states that lead away from the goal and rarely visited on the path to it) are especially suitable for robust decision-time planning techniques. In this case, locally robust policies will foster stable performance while also being computationally more focused than background planning methods. Finally, future work may explore alternative model error bounds and projection operators to further broaden the applicability of robust MCTS to deal with imperfect models for online planning.

ACKNOWLEDGMENTS

This work is part of the project *Flexible Assets Bid Across Markets* (FABAM, project number TEUE117015), funded within the Dutch Topsector Energie / TKI Urban Energy by Rijksdienst voor Ondernemend Nederland (RvO). The authors would like to thank the anonymous reviewers for their constructive feedback and the Dexter Energy B.V. team for their encouragement and support.

REFERENCES

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47 (05 2002), 235–256. https://doi.org/10.1023/A:1013689704352
[2] Richard Bellman. 2010. *Dynamic Programming*. Princeton University Press, USA.
[3] A. Ben-tal and A. Nemirovski. 1998. Robust convex optimization. *Mathematics of Operations Research* 23 (1998), 769–805.
[4] Jose Bernardo and Adrian Smith. 2000. *Bayesian Theory*. Vol. 15. Wiley. https://doi.org/10.2307/2983298
[5] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. 2017. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. *CoRR* abs/1709.07857 (2017). arXiv:1709.07857 http://arxiv.org/abs/1709.07857
[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:1606.01540 [cs.LG]
[7] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4:1 (03 2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810
[8] Olivier Buffet and Douglas Aberdeen. 2005. Robust planning with (L)RTDP. *IJCAI International Joint Conference on Artificial Intelligence*, 1214–1219.
[9] Rémi Coulom. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 72–83.
[10] Karina V. Delgado, Leliane N. de Barros, Daniel B. Dias, and Scott Sanner. 2016. Real-time dynamic programming for Markov decision processes with imprecise probabilities. *Artificial Intelligence* 230 (2016), 192–223. https://doi.org/10.1016/j.artint.2015.09.005
[11] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. 2019. Challenges of Real-World Reinforcement Learning. arXiv:1904.12901 [cs.LG]
[12] Larry Epstein and Martin Schneider. 2003. Recursive multiple-priors. *Journal of Economic Theory* 113, 1 (2003), 1–31. https://EconPapers.repec.org/RePEc:eee:jetheo:v:113:y:2003:i:1:p:1-31
[13] José García and Alvaro Peña. 2018. Robust Optimization: Concepts and Applications. *Nature-inspired Methods for Stochastic, Robust and Dynamic Optimization* (July 2018). https://doi.org/10.5772/intechopen.75381
[14] Garud Iyengar. 2005. Robust Dynamic Programming. *Math. Oper. Res.* 30 (05 2005), 257–280. https://doi.org/10.1287/moor.1040.0129
[15] Le-Minh Kieu, Dong Ngoduy, Nicolas Malleson, and Edward Chung. 2019. A stochastic schedule-following simulation model of bus routes. *Transportmetrica B: Transport Dynamics* 7, 1 (2019), 1588–1610. https://doi.org/10.1080/21680566.2019.1670118 arXiv:https://doi.org/10.1080/21680566.2019.1670118
[16] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, Johannes Fürnkranz, Tobias Scheffer, and Myra

Spiliopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–293.

[17] Erwan Lecarpentier and Emmanuel Rachelson. 2019. Non-Stationary Markov Decision Processes a Worst-Case Approach using Model-Based Reinforcement Learning. *CoRR* abs/1904.10090 (2019). arXiv:1904.10090 http://arxiv.org/abs/1904.10090

[18] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. 2017. AI Safety Gridworlds. *CoRR* abs/1711.09883 (2017). arXiv:1711.09883 http://arxiv.org/abs/1711.09883

[19] Shiau Hong Lim, Huan Xu, and Shie Mannor. 2013. Reinforcement Learning in Robust Markov Decision Processes. In *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/0deb1c54814305ca9ad266f53bc82511-Paper.pdf

[20] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. 2020. A Framework for Reinforcement Learning and Planning. arXiv:2006.15009 [cs.LG]

[21] Satoru Murakami. 1971. A DICHOTOMOUS SEARCH. In *The Operations Research Society of Japan.*

[22] Arnab Nilim and Laurent Ghaoui. 2005. Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *Operations Research* 53 (10 2005), 780–798. https://doi.org/10.1287/opre.1050.0216

[23] Tom Pepels, Mark HM Winands, and Marc Lanctot. 2014. Real-time monte carlo tree search in ms pac-man. *IEEE Transactions on Computational Intelligence and AI in games* 6, 3 (2014), 245–257.

[24] M. Prabavathi and R. Gnanadass. 2018. Electric power bidding model for practical utility system. *Alexandria Engineering Journal* 57, 1 (2018), 277 – 286. https://doi.org/10.1016/j.aej.2016.12.002

[25] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., USA.

[26] Stewart Robinson. 1999. Three sources of simulation inaccuracy (and how to overcome them), Vol. 2. 1701 – 1708 vol.2. https://doi.org/10.1109/WSC.1999.816913

[27] Dario Martin Schafroth. 2010. *Aerodynamics, modeling and control of an autonomous micro helicopter.* Ph.D. Dissertation. ETH Zurich, Zürich. https://doi.org/10.3929/ethz-a-006119546 Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 18901, 2010.

[28] Thiago D. Simão, Romain Laroche, and Rémi Tachet des Combes. 2019. Safe Policy Improvement with an Estimated Baseline Policy. arXiv:1909.05236 [cs.LG]

[29] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[30] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. 2013. Robust Markov Decision Processes. *Mathematics of Operations Research* 38, 1 (2013), 153–183. https://doi.org/10.1287/moor.1120.0566 arXiv:https://doi.org/10.1287/moor.1120.0566

[31] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968