

Pessimistic Iterative Planning with RNNs for Robust POMDPs

Galesloot, Maris F.L.; Suilen, Marnix; Simão, Thiago D.; Carr, Steven; Spaan, Matthijs T.J.; Topcu, Ufuk; Jansen, Nils

DOI

[10.3233/FAIA251391](https://doi.org/10.3233/FAIA251391)

Publication date

2025

Document Version

Final published version

Published in

ECAI 2025 - 28th European Conference on Artificial Intelligence, including 14th Conference on Prestigious Applications of Intelligent Systems, PAIS 2025 - Proceedings

Citation (APA)

Galesloot, M. F. L., Suilen, M., Simão, T. D., Carr, S., Spaan, M. T. J., Topcu, U., & Jansen, N. (2025). Pessimistic Iterative Planning with RNNs for Robust POMDPs. In I. Lynce, N. Murano, M. Vallati, S. Villata, F. Chesani, M. Milano, A. Omicini, & M. Dastani (Eds.), *ECAI 2025 - 28th European Conference on Artificial Intelligence, including 14th Conference on Prestigious Applications of Intelligent Systems, PAIS 2025 - Proceedings* (pp. 4823-4831). (Frontiers in Artificial Intelligence and Applications; Vol. 413). IOS Press. <https://doi.org/10.3233/FAIA251391>

Important note

To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.

Pessimistic Iterative Planning with RNNs for Robust POMDPs

Maris F. L. Galesloot^{a,*}, Marnix Suilen^b, Thiago D. Simão^c, Steven Carr^d, Matthijs T. J. Spaan^e, Ufuk Topcu^d and Nils Jansen^{f,a}

^aRadboud University Nijmegen, NL

^bUniversity of Antwerp – Flanders Make, BE

^cEindhoven University of Technology, NL

^dUniversity of Texas at Austin, US

^eDelft University of Technology, NL

^fRuhr-University Bochum, DE

Abstract. *Robust POMDPs* extend classical POMDPs to incorporate model uncertainty using so-called *uncertainty sets* on the transition and observation functions, effectively defining ranges of probabilities. Policies for robust POMDPs must be (1) memory-based to account for partial observability and (2) robust against model uncertainty to account for the worst-case probability instances from the uncertainty sets. To compute such robust memory-based policies, we propose the *pessimistic iterative planning* (PIP) framework, which alternates between (1) selecting *pessimistic* POMDPs via worst-case probability instances from the uncertainty sets, and (2) computing finite-state controllers (FSCs) for these pessimistic POMDPs. Within PIP, we propose the RFSCNET algorithm, which optimizes a recurrent neural network to compute the FSCs. The empirical evaluation shows that RFSCNET can compute better-performing robust policies than several baselines and a state-of-the-art robust POMDP solver.

1 Introduction

Robust partially observable Markov decision processes (RPOMDPs) extend standard POMDPs with sets of probabilistic transition and observation functions [51]. These *uncertainty sets* account for imprecision that may, for instance, come from probabilities derived from data, sensors with limited precision, or domain experts expressing uncertainty [34, 63]. Such settings are unsuitable for the standard POMDP assumption that probabilities are precisely known.

Policies select actions based on limited state information towards some objective, for instance, optimizing the expected reward. These policies require *memory* as they inherently depend on the sequences of past actions and observations, also known as the *history*.

Robust policies for RPOMDPs must furthermore account for model uncertainty *pessimistically*, that is, they optimize against the *worst-case* instances within the uncertainty sets, providing a lower bound on their actual performance and ensuring their robustness. In summary, RPOMDPs require policies that (1) use memory to reason over the histories and (2) account for model uncertainty.

To robustly optimize a policy against pessimistic instances of the RPOMDP, a policy's *worst-case* performance needs to be assessed

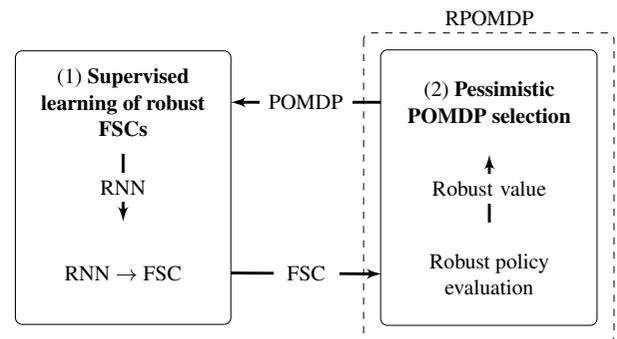


Figure 1. Overview of the PIP framework. The steps on the left, generating an FSC for pessimistic POMDPs through an RNN, are specific to RFSCNET.

analytically. Finite-state controllers (FSCs) are a suitable policy representation for which we can find the worst-case performance analytically through efficient robust dynamic programming [30]. However, computing FSCs often relies on selecting a predetermined memory size and structure [31]. Instead, we desire a more flexible memory structure, which we can achieve by optimizing a *recurrent neural network* (RNN) to search for FSCs [11]. An RNN has a flexible memory structure that can learn sufficient statistics of the histories from data [40]. The drawback, however, is that analytically determining the worst-case performance of an RNN-based policy is intractable.

While optimizing policies for POMDPs is extensively studied [23, 2, 56, 38], methods to compute robust policies for RPOMDPs are sparse, with the most notable approaches being robust point-based value iteration (RPBVI) [51] and sequential convex programming (SCP) [16]. Critically, RPBVI scales poorly to large state spaces, and although SCP scales well, it requires a pre-specified FSC structure, which, as shown in our experimental evaluation, may deteriorate performance. These drawbacks demonstrate the need for flexible and scalable techniques to compute robust policies for RPOMDPs.

Our Approach and Contributions

We present a new approach for computing robust memory-based policies for RPOMDPs. We propose a general framework for plan-

* Corresponding Author. Email: maris.galesloot@ru.nl. The article with appendix is on *arXiv* [19]. Code: <https://doi.org/10.5281/zenodo.16947259>.

ning in RPOMDPs, and by combining the strengths of the flexibility of memory representation of RNNs and the exact robust performance evaluation of FSCs, an algorithmic instance of this framework where we leverage RNNs to compute FSC policies for RPOMDPs. Specifically, our contributions are:

Pessimistic iterative planning for RPOMDPs (Section 4). We present the *pessimistic iterative planning* (PIP) framework to find robust policies for RPOMDPs, as outlined in Figure 1. PIP iteratively computes policies for POMDPs within the uncertainty sets of the RPOMDP that are *pessimistic* instances to the current policy, *i.e.*, a worst-case instance given the current policy. PIP alternates between two main steps: (1) computing FSCs for (pessimistic) POMDPs and (2) evaluating the FSC on the RPOMDP and selecting pessimistic POMDPs. We implement PIP in RFSCNET, an RNN-based algorithm consisting of two parts corresponding to the steps of PIP:

- (1) **Supervised learning of robust FSCs** (Section 5). We train an RNN based on data collected from *supervision policies* that we optimize for the pessimistic POMDPs. From the RNN, we extract the FSC that we use for robust policy evaluation and pessimistic POMDP selection. With these pessimistic POMDPs, we further train the RNN by refining the collected histories and associated supervision policy, guiding the RNN and, therefore, the extracted FSCs towards a robust policy.
- (2) **Pessimistic POMDP selection** (Section 6). First, we compute the worst-case performance of the FSC on the RPOMDP via our robust policy evaluation, thereby producing a guaranteed lower bound on its performance. Using the results from the FSC’s robust evaluation, we construct a linear program that efficiently finds a POMDP instance within the uncertainty sets that is pessimistic, *i.e.*, a worst-case, to the current FSC.

The experimental evaluation on four benchmarks showcases that (1) the FSCs found by RFSCNET are competitive with and in some cases outperform the state-of-the-art SCP solver for RPOMDPs [16], and (2) using PIP increases robustness compared to baselines that train on POMDPs that are heuristically chosen from the uncertainty sets.

2 Preliminaries

The set of all probability distributions over X is denoted by $\Delta(X)$. A distribution $\mu \in \Delta(X)$ is called *Dirac* if $\mu(x) = 1$ for precisely one $x \in X$ and zero otherwise. The number of elements in a set X is denoted by $|X|$. *Iverson brackets* return $[P] = 1$ if predicate P is true and 0 otherwise. Finally, the set of *probability intervals* with lower bounds strictly greater than zero is $\mathbb{I} = \{[i, j] \mid 0 < i \leq j \leq 1\}$.

Definition 1 (POMDP). A partially observable Markov decision process is a tuple $M = \langle S, A, T, Z, O, C \rangle$, where S, A, Z are finite sets of states, actions, and observations, $T: S \times A \rightarrow \Delta(S)$ is the transition function, $O: S \rightarrow Z$ is the deterministic state-based observation function, and $C: S \times A \rightarrow \mathbb{R}_{\geq 0}$ is the cost function.

For simplicity, we consider POMDPs with *deterministic* observations, which is without loss of generality, as every POMDP can be transformed into one with deterministic observations at the cost of a polynomial increase in the size of the state space [13].

A *trajectory* in a POMDP is a sequence of states and actions: $\omega_H = s_1 a_1 s_2 \dots s_H \in (S \times A)^{H-1} \times S$, such that $T(s_{t+1} \mid s_t, a_t) > 0$ for $1 \leq t < H$. A *history* is the observable fragment of a trajectory: $h_H = O(s_1) a_1 O(s_2) \dots O(s_H) = z_1 a_1 z_2 \dots z_H \in (Z \times A)^{H-1} \times Z$. The sets of all trajectories and associated histories

are Ω and \mathcal{H} , respectively. Histories can be summarized into sufficient statistics known as *beliefs*, that is, probability distributions over states [33, 4]. The set of (reachable) beliefs is $\mathcal{B} \subseteq \Delta(S)$. The initial state distribution (belief) is $b_0 \in \mathcal{B}$. A belief $b \in \mathcal{B}$ can be computed from a history $h \in \mathcal{H}$ and b_0 using Bayes’ rule [60].

A *policy* resolves the action choices in a POMDP and is a function $\pi: \mathcal{H} \rightarrow \Delta(A)$ that maps histories to distributions over actions. Since beliefs are sufficient statistics for histories, policies may also be belief-based, *i.e.*, of type $\pi: \mathcal{B} \rightarrow \Delta(A)$. A policy is deterministic if it only maps to Dirac distributions, and the set of all (history-based) policies is denoted by Π .

We focus on minimizing the expected cost of reaching a given set of goal states $G \subseteq S$, also known as the *stochastic shortest path* (SSP) problem [6]. While we focus the presentation of this paper on the SSP problem, it generalizes to discounted rewards [52, 8]. For any trajectory ω , the cumulative cost $\rho_{\diamond G}: \Omega \rightarrow \mathbb{R}_{\geq 0} \cup \{+\infty\}$ is [18]:

$$\rho_{\diamond G}(\omega) = \begin{cases} \infty & \forall t \in \mathbb{N}, s_t \notin G, \\ \sum_{t=0}^{\min\{t \mid s_t \in G\}-1} C(s_t, a_t) & \text{otherwise.} \end{cases}$$

The objective is to find an optimal policy $\pi \in \Pi$ that minimizes the *expected cost* J_T^π of policy π under transition function T :

$$\pi^* \in \operatorname{argmin}_{\pi \in \Pi} J_T^\pi, \quad \text{where} \quad J_T^\pi = \mathbb{E}_{\pi, T}[\rho_{\diamond G}(\omega) \mid s_0 \sim b_0].$$

Here, the expectation $\mathbb{E}_{\pi, T}[\cdot]$ is over the trajectories ω generated by following policy π under the transition function T . The decision variant of the problem of finding an optimal policy for the SSP problem in a POMDP is undecidable [42]. Therefore, it is common to approximate optimal policies with finite memory. A policy is *finite-memory* if it can be represented by a finite-state controller (FSC) [23].

Definition 2 (FSC). A finite-state controller for a POMDP M is a tuple $\pi_f = \langle N, n_0, \delta, \eta \rangle$ where N is a finite set of memory nodes, $n_0 \in N$ the initial node, $\delta: N \times Z \rightarrow \Delta(A)$ is the action function, and $\eta: N \times Z \rightarrow N$ is the memory update.

$\Pi_f \subseteq \Pi$ denotes the set of FSCs. At execution time, at state s and node n , using $z = O(s)$, the FSC selects action $a \sim \delta(\cdot \mid n, z)$ and updates its node to $n' = \eta(n, z)$. By $\psi(a, n' \mid s, n) = \delta(a \mid n, O(s))[n' = \eta(n, O(s))]$, we denote the joint probability of the FSC selecting action a and updating to the next memory node n' . The expected costs $J_T^{\pi_f}$ of an FSC π_f on a POMDP M is evaluated by computing the state-values on the product Markov chain [43, 5].

3 Robust POMDPs

Robust POMDPs (RPOMDPs) [51, 9] extend POMDPs by accounting for uncertainty in the transition and observation functions. That is, the probabilities are no longer given, but only known to belong to some *uncertainty set* [67, 51]. Without loss of generality, we focus on uncertainty over the transition function, as any RPOMDP with an uncertain observation function can be transformed into an equivalent RPOMDP with a deterministic observation function, see Appendix B [19] or [9]. Similarly, we omit reward uncertainty for brevity, but it can be included in a straightforward manner [51].

Definition 3 (RPOMDP). A robust POMDP with interval uncertainty is a tuple $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O \rangle$, where S, A, Z , and C, O are as in Definition 1, and $\mathcal{T}: S \times A \rightarrow (S \rightarrow \mathbb{I} \cup \{0\})$ is the uncertain transition function that maps transitions to either a probability interval in \mathbb{I} , or probability 0 whenever the transition does not exist.

Henceforth, we assume the standard rules for interval arithmetic [26]. We allow only intervals with a lower bound greater than zero to avoid the vanishing of transitions, also known as *graph preservation*. This assumption is standard if intervals are learned from data [67]. Furthermore, for the objective considered here, a lower bound of zero is only possible if one assumes all trajectories reach the goal states G with probability one. Otherwise, G may become unreachable in the worst case.

A robust MDP (RMDP) [62] is a fully observable RPOMDP, *i.e.*, where $Z \equiv S$ and $\forall s \in S: O(s) = s$. A POMDP M with transition function T is called an *instance* of the RPOMDP if every transition probability of T lies within its respective interval in \mathcal{T} . With abuse of notation, we may also write $T \in \mathcal{T}$ and $M \in \mathcal{M}$.

RPOMDPs can be seen as a game between the agent and a second player, *nature*, which selects probability distributions from the uncertainty set. We assume the *dynamic* uncertainty model [30], which means that nature’s choices are not restricted by any previous choice, and independence between all state-action pairs, a common assumption known as (s, a) -*rectangularity* [67]. Specifically, the uncertain transition function \mathcal{T} factorizes over state-action-pairs, *i.e.*, it is comprised of the Cartesian product $\mathcal{T} = \bigotimes_{(s,a) \in S \times A} \mathcal{T}(s, a)$, where:

$$\mathcal{T}(s, a) = \{T(s, a) \in \Delta(S) \mid \forall s' \in S: T(s' \mid s, a) \in \mathcal{T}(s, a)(s')\}.$$

While we focus on uncertainty in the form of intervals, the results presented in this paper generalize to (s, a) -rectangular uncertainty sets that are graph preserving and form *convex polytopes*, such as those constructed from the ℓ_1 or ℓ_∞ norms.

RPOMDPs have two optimal value functions and associated optimal policies: one where the agent and nature play *adversarially*, and one where they play cooperatively. The former is the robust (or pessimistic) setting, and the latter is the optimistic setting. Our approach extends to both the robust and the optimistic case. We focus the presentation on the robust setting in the remainder of this paper.

In RPOMDPs, the trajectories generated by a policy $\pi \in \Pi$ depend on the transition function $T \in \mathcal{T}$. Hence, we minimize the *robust value* \mathcal{J}_T^π , which represents the *worst-case expected cost*:

$$\pi^* \in \operatorname{arginf}_{\pi \in \Pi} \mathcal{J}_T^\pi, \quad \text{where} \quad \mathcal{J}_T^\pi = \sup_{T \in \mathcal{T}} J_T^\pi.$$

The undecidability of the decision variant of this problem follows from that of our objective in POMDPs [42]. Therefore, we cannot aim for completeness. Instead, we focus on developing a practical and modular algorithm that computes robust FSCs, including a sound policy evaluation, while allowing for a flexible memory structure.

Goal. Given an RPOMDP $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O \rangle$, compute an FSC $\pi_f \in \Pi_f$ that minimizes the robust value $\mathcal{J}_T^{\pi_f}$.

4 Pessimistic Iterative Planning

We present our main contributions. First, we outline the two main parts of the *pessimistic iterative planning* (PIP) framework. Subsequently, we give an overview of our algorithmic implementation of PIP, named RFSCNET, which computes robust FSCs for RPOMDPs by optimizing an RNN and extracting FSCs from the RNN.

4.1 The PIP Framework

Analogously to a sequential two-player game, PIP iteratively executes two parts, representing the two sides of Figure 1:

(1) Compute an FSC policy π_f for a given POMDP $M \in \mathcal{M}$.

(2) Select a *pessimistic* POMDP $\underline{M} \in \mathcal{M}$ with respect to π_f , and set $M \leftarrow \underline{M}$. Give M as input to (1).

These steps are repeated in a sequential game-like fashion. The first player executes Step (1) and computes an FSC π_f that minimizes the expected cost in the (non-robust) POMDP M . Note that PIP may use any existing approach that computes FSCs for (non-robust) POMDPs. The other player then executes Step (2) and determines a POMDP $\underline{M} \in \mathcal{M}$ that is pessimistic with respect to π_f , effectively *maximizing* the expected cost incurred under π_f . Returning to Step (1), the first player takes \underline{M} into account and computes a new (updated) FSC optimized for \underline{M} . This process is repeated until we reach a *termination criterion*, *i.e.*, the robust performance of the FSC satisfies a target threshold or a maximum number of iterations is reached.

On convergence. Computing robust policies for cost minimization in an RPOMDP results in a sequential zero-sum game between the agent and nature [27, 9]. PIP approximates this game through an iterated best-response formulation, where both the agent and nature optimize their policy based on the given policy of the other player. The goal of this paper is not to formalize the resulting game. Still, we present our considerations in the following. In finite games where each player optimizes over a finite set of policies, convergence is achieved when the optimization of both players’ policies reaches a saddle point, *i.e.*, $\inf_{\pi_f \in \Pi_f} \sup_{T \in \mathcal{T}} J_T^{\pi_f} = \sup_{T \in \mathcal{T}} \inf_{\pi_f \in \Pi_f} J_T^{\pi_f}$, and a Nash equilibrium follows under mild conditions [54]. If we limit the agent to deterministic FSCs, it can be argued that the set of policies of both players is finite. However, the sequential structure of the PIP approximation does not fit directly into the classical Nash formulation. Stackelberg games encapsulate sequential games [69, 54], but assume that the first player considers all possible responses of the other player, which does not occur in PIP. Establishing whether equilibria exist and whether a formal guarantee of convergence for a framework like PIP is possible is an open problem. Therefore, PIP does not aim at completeness; we cannot guarantee that an optimal policy or a set threshold will be achieved in the long run of PIP. Instead, the PIP framework provides soundness: the robust policy evaluation steps provide a bound on the worst-case performance. Together with a preset threshold, this evaluation provides a sound termination criterion and an additional limit to the number of iterations.

4.2 The RFSCNET Algorithm

Next, we detail the steps of our algorithm RFSCNET, which implements the two parts of PIP. Part one corresponds with Section 5 and the left-hand side in Figure 1. We compute FSCs for input POMDPs $M \in \mathcal{M}$ using an RNN, which is specific to RFSCNET:

- i Compute a *supervision policy* π_M for the input POMDP M (Section 5.1) and simulate π_M on M to collect the histories and the action distributions of π_M into a data set \mathcal{D} (Section 5.2).
- ii Train the RNN policy π^ϕ on the data set \mathcal{D} (Section 5.3) and extract an FSC π_f (Section 5.4).

Part two corresponds to Section 6 and the right-hand side of Figure 1. Here, we implement the robust policy evaluation of a computed FSC and, subsequently, the selection of new pessimistic POMDPs. These steps other ways of computing the FSC. In this part, we:

- iii Evaluate the robust value $\mathcal{J}_T^{\pi_f}$ of the FSC π_f through robust dynamic programming (Section 6.1).
- iv Compute a new pessimistic POMDP $\underline{M} \in \mathcal{M}$ based on the FSC π_f (Section 6.2). Set $M \leftarrow \underline{M}$ as the next input.

We track the best policy found based on $\mathcal{J}_{\tau}^{\pi^f}$, and we determine whether to stop the algorithm based on the aforementioned termination criteria of the PIP framework. Otherwise, after Step (iv), we start a new iteration at Step (i) using \underline{M} as the new input POMDP.

We opt for a model-based approach, using approximate solvers to compute *supervision policies* π_M for $M \in \mathcal{M}$ and train the RNN to imitate π_M in a supervised manner, resembling *imitation learning* [57, 28]. Alternatively, we could employ model-free techniques from reinforcement learning, such as *recurrent policy gradients* [66], to optimize the RNN. Our method readily supports this. However, model-free methods neglect the available information from the model. Therefore, in addition to the fact that reinforcement learning under function approximation may diverge [17], it typically requires many samples to achieve reasonable values [55, 44].

5 Supervised Learning of Robust FSCs

In this section, we specify the methods in RFSCNET used to compute FSCs given input POMDPs: collecting data, RNN architecture and training, and extracting a finite-state controller policy from an RNN.

5.1 Supervision Policies

Since computing an optimal policy in POMDPs for our objective is undecidable (recall Section 2) and our framework relies on computing policies for multiple POMDPs, fast approximate methods are needed. We compute *supervision policies* $\pi_M: \mathcal{B} \rightarrow \Delta(A)$ that approximate the optimal policy π^* for the POMDP M . In particular, we approximate the belief-based action-values $Q_M: \mathcal{B} \times A \rightarrow \mathbb{R}$ for $M \in \mathcal{M}$, which we denote by \hat{Q}_M , and use either Q_{MDP} [41] or the fast-informed bound (FIB) [24] to compute these approximations, denoted Q_{MDP} and Q_{FIB} respectively. The supervision policy is then derived by acting greedily, *i.e.*, taking a Dirac distribution on the minimizing action, such that $\pi_M(a | b) = [a = \arg\min_{a \in A} \hat{Q}_M(b, a)]$.

Q_{MDP} neglects information-gathering and assumes full state observability after a single step [41]. The values Q_{FIB} are tighter than those of Q_{MDP} since it factors in the observation of the next state. See Appendix C [19] for more details. Other methods may also be used, such as variants of *partially observable Monte Carlo planning* [58] or *heuristic-search value iteration* [59] for POMDPs with target states.

5.2 Data Generation

To train the RNN, we generate a dataset \mathcal{D} by simulating the supervision policy π_M on the current (pessimistic) POMDP $M \in \mathcal{M}$. We execute $I \in \mathbb{N}$ simulations up to length $H \in \mathbb{N}$. During the simulations, we play the actions of the supervision policy π_M . We aggregate the histories and associated action distributions generated by following π_M .

To elaborate, at each time step $1 \leq t < H$ of simulation i , we track the beliefs $b_t^{(i)}$ associated with history $h_t^{(i)}$, using the transition function T of M , with $b_0^{(i)}$ the initial belief b_0 of the RPOMDP M . Then, $\mu_t^{(i)} = \pi_M(\cdot | b_t^{(i)})$ is the action distribution of the supervision policy π_M during the simulation, and $a_t^{(i)} \sim \mu_t^{(i)}$ is the action used in simulation i at time t . We store the histories and action distributions in the dataset $\mathcal{D} = \{\{h_t^{(i)}, \mu_t^{(i)}\}_{t=1}^H\}_{i=1}^I$, which then consists of $I \cdot H$ histories and the associated action distributions. We only consider the data generated by simulation in the most recent iteration, which we found leads to the most stable results. In the next step, we employ \mathcal{D} to train the RNN policy π^ϕ .

5.3 RNN Policy Architecture and Training

Comparably to FSCs, RNNs are (infinite) state machines parameterized by differentiable functions. The states $\hat{h} \in \hat{\mathcal{H}} \subseteq \mathbb{R}^d$ represent memory, where d defines the *hidden size* of the vector. Analogously to an FSC, we design the RNN to consist of a parameterized internal memory update $\hat{\eta}_\phi: \hat{\mathcal{H}} \times Z \rightarrow \hat{\mathcal{H}}$ that recurrently computes the new latent state $\hat{h} \in \hat{\mathcal{H}}$ from the observations of a history $h \in \mathcal{H}$. Thus, the RNN represents a function RNN: $\mathcal{H} \rightarrow \hat{\mathcal{H}}$. When we append a fully connected layer with a softmax activation function $\sigma^\phi: \hat{\mathcal{H}} \rightarrow \Delta(A)$ to the RNN, it yields an RNN policy network $\pi^\phi: \mathcal{H} \rightarrow \Delta(A)$ from histories to distributions over actions.

The training objective for the RNN is to minimize the distance between the distributions of the RNN policy π^ϕ and the distributions μ of the supervision policy π_M over the dataset \mathcal{D} : $\min_\phi \frac{1}{|\mathcal{D}|H} \sum_{i=1}^{|\mathcal{D}|} \sum_{t=1}^H L(\pi^\phi(h_t^{(i)}) \parallel \mu_t^{(i)})$, where L is a distance or loss function, *e.g.*, Kullback-Leibler divergence, and for each batch index i and time-step t , the histories $h_t^{(i)}$ are the RNN's inputs, and the action distributions $\mu_t^{(i)}$ are the labels. To optimize the parameters ϕ of the RNN, we calculate the gradient via backpropagation through time [65]. See Appendix D.2 [19] for more details.

5.4 Extracting an FSC from an RNN

Recall that in our approach, we change the POMDP at each iteration to be pessimistic against the current policy. For robust policy evaluation and to select the new pessimistic POMDP $\underline{M} \in \mathcal{M}$, we need a finite-memory representation of the policy, which we find as follows.

We cluster the hidden memory states of the RNN [68, 50] to find an FSC. Prior work [11] uses a quantized bottleneck network (QBN) [37] to reduce the possible memory states to a finite set. They train the QBN *post hoc* by minimizing the mean-squared error on the hidden states generated by the RNN. Alternatively, we can train it *end-to-end* by updating its parameters with the loss from Section 5.3, which we name QRNN for quantized RNN. Moreover, similar to post hoc training of the QBN, we can run a clustering algorithm such as *k-means++* [3] to minimize the in-cluster variance of the hidden states. For post hoc training, we employ the histories in \mathcal{D} to generate the RNN hidden states. We consider all three methods in our experiments and provide more details of the QBN in Appendix D.1 [19].

Instead of through simulations, as done by Koul et al. [37], we utilize the model to construct the FSC. The clustering determines a finite set N of memory nodes of the RNN's possible hidden states, *i.e.*, a partitioning of $\hat{\mathcal{H}}$. We find the FSC's memory update η by executing a forward pass of the RNN's memory update $\hat{\eta}_\phi$ for each reconstruction of $n \in N$, which produces the next memory nodes $n' \in N$ and RNN hidden state $\hat{h} \in \mathcal{H}$ for each $z \in Z$ by exploiting the batch dimension. Then, the action mapping δ for n and z is given by the distribution of the RNN policy network $\sigma^\phi(\hat{h})$ for the next memory state. The RNN's initial state determines the initial node n_0 , and we prune any unreachable nodes from the FSC.

Alternatives include unfolding π^ϕ and minimizing the resulting policy tree [21], or using active automata learning [45]. Yet, these methods do not adequately scale since the size of the policy tree is exponential in the horizon, which is an inherent problem of POMDPs.

6 Robust Policy Evaluation and Pessimistic Selection of POMDPs

The previous section explains how we find FSCs given the input pessimistic POMDPs. Now, we present our methods for a sound robust

policy evaluation of the FSC and, subsequently, for selecting a pessimistic, *i.e.*, a worst-case POMDP $\underline{M} \in \mathcal{M}$ from this evaluation.

6.1 Robust Policy Evaluation

We evaluate the robust performance of the FSC on the RPOMDP via the product construction of a *robust Markov chain* (RMC), similar to the one used for evaluating FSCs in (non-robust) POMDPs [43].

Definition 4 (RMC). *From an RPOMDP $\mathcal{M} = \langle S, A, \mathcal{T}, C, Z, O \rangle$ with initial belief b_0 and an FSC $\pi_f = \langle N, n_0, \delta, \eta \rangle$ we construct a robust Markov chain $\mathcal{M}^{\pi_f} = \langle S \times N, b_0^{\pi_f}, \mathcal{P}^{\pi_f}, C^{\pi_f} \rangle$ where the state-space is the product of RPOMDP states S and FSC memory nodes N , the initial state distribution is $b_0^{\pi_f}(\langle s, n \rangle) = b_0(s)[n = n_0]$. The uncertain transition and cost functions are as follows:*

$$\begin{aligned} \mathcal{P}^{\pi_f}(\langle s', n' \rangle | \langle s, n \rangle) &= \sum_{a \in A} \mathcal{T}(s, a)(s')\psi(a, n' | s, n), \\ C^{\pi_f}(\langle s, n \rangle) &= \sum_{a \in A} \delta(a | n, O(s))C(s, a). \end{aligned}$$

The robust value $\mathcal{J}_T^{\pi_f}$ of the FSC π_f is determined by computing the robust state-value $\underline{V}^{\pi_f}: S \times N \rightarrow \mathbb{R}_{\geq 0}$ of the RMC \mathcal{M}^{π_f} via dynamic programming [53, 30], such that its value $\mathcal{J}_T^{\pi_f} = \sum_{s, n} b_0^{\pi_f}(\langle s, n \rangle) \underline{V}^{\pi_f}(\langle s, n \rangle)$ is the worst-case expected cost of the FSC given the assumptions of this paper. The robust state-value \underline{V}^{π_f} is the fixed point [53] of the following robust Bellman equation:

$$\begin{aligned} \underline{V}^{\pi_f}(\langle s, n \rangle) &= C^{\pi_f}(\langle s, n \rangle) + \sup_{P^{\pi_f}(\langle s, n \rangle) \in \mathcal{P}^{\pi_f}(\langle s, n \rangle)} \\ &\sum_{s' \in S} \sum_{n' \in N} P^{\pi_f}(\langle s', n' \rangle | \langle s, n \rangle) \underline{V}^{\pi_f}(\langle s', n' \rangle). \end{aligned} \quad (1)$$

where $\mathcal{P}^{\pi_f}(\langle s, n \rangle) = \sum_{a \in A} \mathcal{T}(s, a)\psi(a, \cdot | s, n)$ is the resulting uncertainty set in the robust Markov chain. If the FSC π_f reaches the target set G with probability one, then the above Bellman equation is a contraction with respect to a weighted maximum norm [7, 52, 53].

Effectively, (s, a) -rectangularity carries over to the product state-space of the RMC, leading to $(\langle s, n \rangle, a)$ -rectangularity; the uncertainty set additionally factorizes over the memory nodes. That is, nature's choices can change depending on the agent's memory node, yielding a conservative bound on the robust value, as formalized in the following theorem.

Theorem 1. *Given the FSC π_f , the robust state-values \underline{V}^{π_f} of the RMC \mathcal{M}^{π_f} provide a conservative (upper) bound on the value $\underline{V}_*^{\pi_f}$ of the FSC π_f under (s, a) -rectangularity in the RPOMDP.*

Since the agent policy π_f is fixed, dynamic programming on the RMC is tractable due to both the rectangularity and convexity of the uncertainty sets, and the dynamic uncertainty model [49, 30, 14].

We could consider the $(\langle s, n \rangle, a)$ -rectangularity simply as (s, a) -rectangularity plus the dynamic uncertainty model, allowing nature to choose different probabilities when revisiting $(s, a) \in S \times A$ for different $n \in N$. However, the dynamic model in this case does not converge to a fixed probability distribution, *i.e.*, a static model, for each (s, a) pair as is the case for RMDPs with stationary policies [30]. Instead, it converges to a fixed probability distribution for each (s, n, a) triplet. It follows from the fact that a static model results in the same value as a dynamic uncertainty model over the state space of the robust Markov chain [30, 14]. The intuition here is that the FSC π_f is a stationary policy over $S \times N$. Therefore, it also suffices for nature to act with a stationary policy. The next theorem formalizes this result.

Theorem 2. *In the RMC \mathcal{M}^{π_f} , the value under the dynamic model \underline{V}^{π_f} coincides with the value under a static model $\hat{\underline{V}}^{\pi_f}$.*

Appendix A [19] contains the proofs for the two theorems above. Thus, there exists a transition function $T_N: S \times N \times A \rightarrow \Delta(S)$ that is adversarially chosen by nature that induces the worst-case expected costs of the FSC π_f . In addition to the usual transition function of a POMDP, it also depends on the memory update function of the FSC.

6.2 Selecting Pessimistic POMDP Instances

We now construct a heuristic to find a new POMDP instance $\underline{M} \in \mathcal{M}$ that constitutes a local worst-case instance for the current policy $\pi_f \in \Pi_f$ under (s, a) -rectangularity of the RPOMDP.

Let $\pi_f = \langle N, n_0, \delta, \eta \rangle$ be the current FSC, and let $P_T^{\pi_f} \in \mathcal{P}^{\pi_f}$ denote the transition function of the Markov chain by selecting the transition probabilities $T \in \mathcal{T}$. Given the robust value function \underline{V}^{π_f} computed from Equation (1), we aim to find a POMDP $\underline{M} \in \mathcal{M}$ that induces its worst-case value and, thus, is pessimistic to π_f .

Definition 5 (Pessimistic POMDP). *Given an FSC policy π_f and its robust value function \underline{V}^{π_f} , a pessimistic POMDP $\underline{M} \in \mathcal{M}$ is a POMDP $\underline{M} = \langle S, A, \underline{T}, C, Z, O \rangle$ with a pessimistic transition function $\underline{T} \in \mathcal{T}$ with respect to the robust value function \underline{V}^{π_f} , such that $\underline{T} \in \operatorname{argmax}_{T \in \mathcal{T}} P_T^{\pi_f} \underline{V}^{\pi_f}$.*

As described in the previous subsection and in more detail in Appendix A [19], the robust value function is effectively computed under $(\langle s, n \rangle, a)$ -rectangularity, resulting in probabilities that may differ for each memory node n , *i.e.*, with a transition function of type $T_N: S \times N \times A \rightarrow \Delta(S)$. However, we desire our pessimistic POMDPs to have a typical transition function of the signature $\underline{T}: S \times A \rightarrow \Delta(S)$. Therefore, we compute transition probabilities under the additional constraint that probabilities are independent of the memory nodes $n \in N$, *i.e.*, under (s, a) -rectangularity:

$$\operatorname{argmax}_{T(s, a) \in \mathcal{T}(s, a)} \sum_{n, s', n'} T(s' | s, a)\psi(a, n' | s, n) \underline{V}^{\pi_f}(\langle s', n' \rangle).$$

We construct a single linear program (LP) that precisely encodes our requirements, using the independence among state-action pairs. Let $\hat{T}_{s, a, s'}$ be the optimization variables representing the probabilities of a pessimistic transition function. The LP is formulated as:

$$\begin{aligned} \max_{\hat{T}} \quad & \sum_{s, n, a, s', n'} \hat{T}_{s, a, s'} \psi(a, n' | n, s) \underline{V}^{\pi_f}(\langle s', n' \rangle) \\ \text{s.t.} \quad & \forall s, a \in S \times A: \sum_{s' \in S} \hat{T}_{s, a, s'} = 1, \\ & \forall s, a, s' \in S \times A \times S: \hat{T}_{s, a, s'} \in \mathcal{T}(s, a)(s'). \end{aligned} \quad (2)$$

Solving this LP yields assignments for the variables $\hat{T}_{s, a, s'}$ that determine a *pessimistic transition function* $\hat{\underline{T}}: S \times A \rightarrow \Delta(S)$ that satisfies $\hat{\underline{T}}(s, a) \in \mathcal{T}(s, a)$ for all $(s, a) \in S \times A$. By construction, the assignments are valid probability distributions within each respective interval and yield a heuristic for the worst possible value for the given FSC under (s, a) -rectangularity.

With the selection of a pessimistic POMDP instance $\underline{M} = \langle S, A, \hat{\underline{T}}, C, Z, O \rangle$, we have closed the loop of PIP, and resume the algorithm by optimizing the FSC for the next input $\underline{M} \in \mathcal{M}$ unless we have reached a termination criterion specified in Section 4.1.

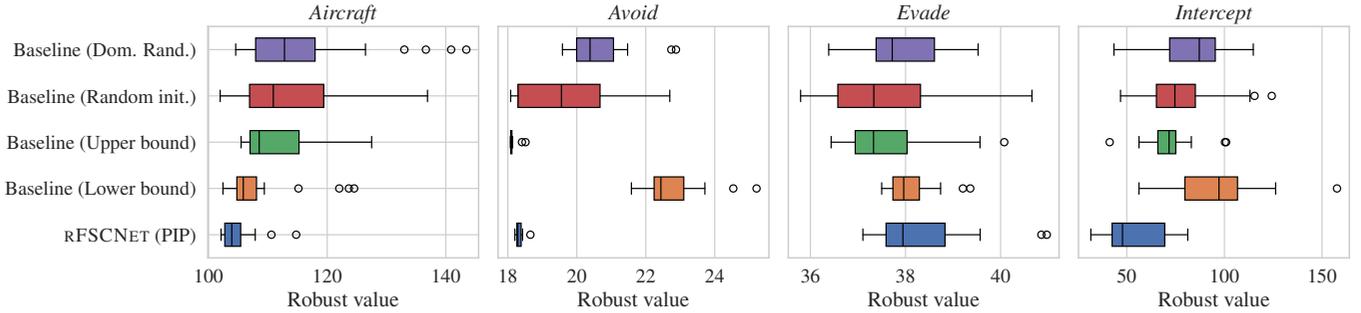


Figure 2. Boxplots depicting robust values (lower is better) of the extracted FSC policies for both RFSCNET and the baselines reported across 20 seeds. All are configured with Q_{MDP} and k -means with $|N| \leq 9$. Generally, RFSCNET finds the most robust policies across environments, in contrast to the algorithmically similar baselines, i.e., they employ an RNN to compute FSCs, but do not employ the pessimistic POMDP selection steps of the PIP framework.

7 Experimental Evaluation

We empirically assess different aspects of RFSCNET to address the following questions:

- (Q1) **Robustness and baseline comparison.** How does RFSCNET compare to various baselines that do not utilize the PIP framework: does PIP enable robust performance?
- (Q2) **Comparison with the state-of-the-art.** How does RFSCNET’s performance compare to the SCP solver?
- (Q3) **Memory size sensitivity.** How does the specified memory size affect RFSCNET and SCP’s performance?
- (Q4) **Configuration sensitivity.** How do different configurations of supervision policies and clustering methods affect the performance of RFSCNET?

The experimental evaluation is set up as follows.

Environments. We use the existing RPOMDP benchmark of the *Aircraft* collision avoidance problem [36, 16], and extend three POMDP grid-worlds with adversaries to RPOMDPs, named *Avoid*, *Evade*, and *Intercept* [32]. On *Aircraft*, the agent is tasked to avoid a collision with an intruder aircraft while accounting for uncertainty in the probabilities of the pilot’s responsiveness and the intruder changing direction, both mapping to a $[0.6, 0.8]$ interval. The grid-world environments model the probability of taking multiple steps instead of a single one for each possible moving action, given by the interval $[0.1, 0.4]$. We report dimensions in Table 1 and provide environment descriptions and run times in Appendices E and G [19], respectively.

Baselines. We evaluate the impact of the pessimistic selection of POMDPs of the PIP framework on robust values against several baselines. These run, like RFSCNET, on POMDPs within the uncertainty sets of the RPOMDP, but selected by one of the following heuristics:

Lower bound. At the first iteration, select a POMDP within the RPOMDP that greedily assigns the lower bounds of the intervals to transitions, and distribute the remaining probability mass to ensure a valid probability distribution. Remains fixed over iterations.

Upper bound. Similar to the above, but greedily assign the upper bounds of the intervals to transitions while ensuring a probability mass less than or equal to one, until it is no longer possible. The remaining probability mass is distributed uniformly over the remaining transitions. It remains fixed once initialized.

Random init. At initialization, randomly select a single POMDP within the RPOMDP that remains fixed throughout the iterations.

Instances	<i>Aircraft</i>	<i>Avoid</i>	<i>Evade</i>	<i>Intercept</i>
$ S $	13552	10225	4261	4802
$ Z $	37	6968	2202	2062
$ A $	5	4	5	4

Table 1. Dimensions of each benchmark environment.

Dom. Rand. At each iteration, randomly select a new POMDP within the RPOMDP, resembling *domain randomization* [64].

At each iteration, the POMDP used is described above. The baselines and RFSCNET run for the same number of 50 iterations.

Metric. We compare the *robust values* $\mathcal{J}_{\mathcal{T}}^{\pi_f}$, of the FSCs π_f computed by the baselines, RFSCNET, and SCP. For RFSCNET and the baselines, we consider the best robust value found across the iterations. As these methods include randomness in the sampling, initialization, and training, we report statistics of the robust value across 20 seeds. SCP is not random given a fixed initialization of its hyperparameters, thus we only report a single value for its default settings.

Tools and Hyperparameters. We use the tools Storm [25] for parsing the models and PRISM [39] to compute the RMDP values for the lower bounds in Table 2 and for robust value iteration on the robust Markov chain in robust policy evaluation. We build and train the RNN and the QBN using TensorFlow [1]. The RNN cell is a gated recurrent unit (GRU) [15]. For all the experiments, the simulation batch size is set to $I = 256$, the maximum simulation length is set to $H = 200$, and we run for a maximum of 50 iterations. The RNN and QBN use an Adam optimizer [35] with a learning rate of $1 \cdot 10^{-3}$. The hidden size of the RNNs was set to $d = 16$. For solving the LP, we use the Gurobi solver [22]. The experiments ran inside a Docker container on a Debian 12 machine. Our infrastructure includes an AMD Ryzen Threadripper PRO 5965WX machine with 512 GB of RAM. We train the neural networks on the CPU. The different seeds for the RNN-based methods were executed in parallel, each running on a single core. Multi-threading in the Gurobi solver used by SCP was enabled. In our initial tests, we considered hidden sizes $d \in \{3, 16, 64\}$, batch sizes $I \in \{128, 256, 512\}$, learning rates in the range of $[1 \cdot 10^{-2}, 1 \cdot 10^{-4}]$, and different number of iterations before arriving at our final values. We used the same infrastructure and experimental setup across methods.

Results. Figure 2 compares the performance of RFSCNET to the aforementioned baselines. Table 2 shows RFSCNET’s median and

Robust Values $\mathcal{J}_T^{\pi^f}$		<i>Aircraft</i>	<i>Avoid</i>	<i>Evade</i>	<i>Intercept</i>
Lower Bound (RMDP)		94.24	18.05	31.19	16.99
SCP	($ N = 3$)	116.03	20.07	37.97	31.57
	($ N = 9$)	116.58	29.51	39.78	101.12
RFSCNET	(med.)	103.91	18.29	37.95	47.82
	(min.)	102.10	18.20	37.10	31.61
	(iqr.)	± 2.69	± 0.12	± 1.24	± 26.97

Table 2. The median (med.), minimum (min.), and interquartile range (iqr.) of the robust values of RFSCNET across 20 seeds compared to those values of SCP (lower is better). RMDP denotes a lower bound on the robust value by computing the robust value of the underlying RMDP, assuming full observability. We highlight the best value between SCP and the med. of RFSCNET.

minimum performance when configured with a maximal memory size of $|N| = 9$, compared to the SCP method with two different sizes $|N| \in \{3, 9\}$. The heatmaps in Figure 3 showcase the effect of various memory sizes $|N|$ on the performance of both RFSCNET and SCP in *Aircraft* and *Evade*. In these results, RFSCNET is equipped with `k-means++` clustering and Q_{MDP} as supervision policy. Figure 4 shows the difference in training performance, in terms of the RNN and QBN losses, between RFSCNET and a baseline on *Intercept*. Figure 5 compares RFSCNET across multiple supervision policies and clustering configurations to a baseline on *Intercept*. In Appendix F [19], we provide the complete set of results.

7.1 Analysis

We now address each research question based on the experiments.

(Q1) Comparison to baselines. As seen in Figure 2, RFSCNET outperforms all baselines on *Aircraft* and *Intercept*, reaching a median value that is lower than the first quartile of all baselines. On average, RFSCNET incurs lower expected costs than the baselines. Figure 4 shows that in *Intercept*, training on pessimistic POMDPs, instead of a single fixed POMDP, is a more difficult learning target. Even though the training task is more challenging, RFSCNET still performs well. The results are more ambiguous in the *Evade* environment. The baselines can perform better, demonstrating that ignoring the model uncertainty may suffice in this environment. Nonetheless, we still observe that RFSCNET achieves at least the same robust performance as the baselines. On *Avoid*, RFSCNET performs slightly worse than the baseline that is trained on the upper bound of the uncertainty set, while the remaining baselines clearly perform very poorly. We conjecture that, by coincidence, the upper bounds provide an adequate approximation of the worst-case probabilities. Without this particular initialization, RFSCNET still achieves comparable performance to this baseline. Good baseline performance is not guaranteed, and the baselines may find much worse performing policies, as evidenced by the results obtained when trained on the lower bound or through domain randomization. Thus, the baselines are unreliable as they are sensitive to the POMDPs used throughout training, while RFSCNET performs reliably across environments.

(Q2) Comparison with the state-of-the-art. As seen in Table 2, in comparison to RFSCNET, SCP performs comparably on *Evade* and best on *Intercept* when $|N| = 3$. These results showcase that SCP performs well if the memory is cherry-picked for the problems. In the case that the memory size in SCP is set to $|N| = 9$, which is the same maximal memory size we set for RFSCNET in these results, RFSCNET significantly outperforms SCP across all benchmarks. Furthermore, RFSCNET significantly outperforms SCP on *Aircraft* and

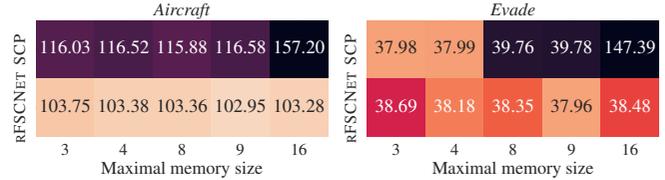


Figure 3. Heatmaps comparing the robust values $\mathcal{J}_T^{\pi^f}$ under various memory sizes $|N|$ of RFSCNET (median across 20 seeds reported) to SCP's. Brighter colors (lower values) are better. With high memory sizes $|N|$, SCP performs worse, whereas RFSCNET performs consistently across all sizes.

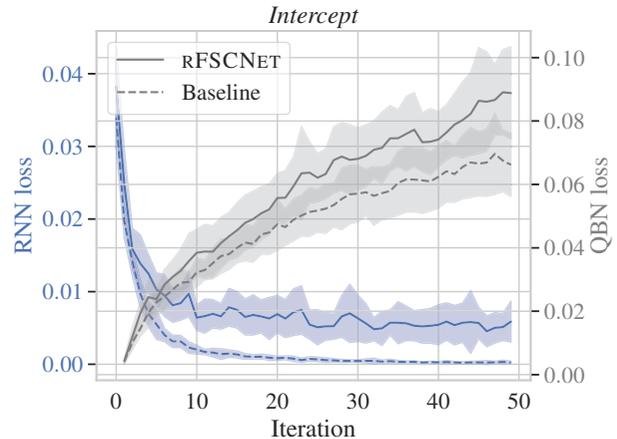


Figure 4. Comparing the *post hoc* QBN and RNN training losses (averaged over the seeds) between a baseline running on a fixed POMDP and RFSCNET on the RPOMDP on *Intercept*. Both are equipped with Q_{MDP} and a QBN.

Avoid across both memory configurations. Therefore, we conclude that RFSCNET improves over the state-of-the-art in these cases.

(Q3) Memory size sensitivity. Table 2 indicates that SCP performs worse with more memory, especially on *Avoid* and *Intercept*. To investigate further, we conduct additional experiments to compare the performance of SCP and RFSCNET on *Aircraft* and *Evade* with increasing memory sizes. The heatmaps in Figure 3 show that SCP also performs much worse on *Aircraft* and *Evade* when more memory is specified. A larger memory size implies more optimization variables for SCP, which may be why its performance deteriorates, as SCP may get stuck in worse local optima when we allow for more memory. In contrast, RFSCNET is not sensitive to specifying more memory nodes than necessary, exhibiting relatively consistent performance across the memory sizes. These results demonstrate the benefit of learning the memory structure instead of specifying it beforehand, as done in SCP.

(Q4) Configuration sensitivity. Figure 5 depicts the performance of RFSCNET across various configurations on *Intercept*. Figure 6 in Appendix F.1 [19] shows these results also for the other three environments. From our results, we did not observe a major difference between using Q_{MDP} and Q_{FIB} as supervision policies. While sometimes producing better results, training the QBN *end-to-end* proves less stable than *post hoc*. Overall, clustering with `k-means++` produces the best results in our benchmarks. The results demonstrate that, while the configuration does impact performance, RFSCNET performs consistently across configurations.

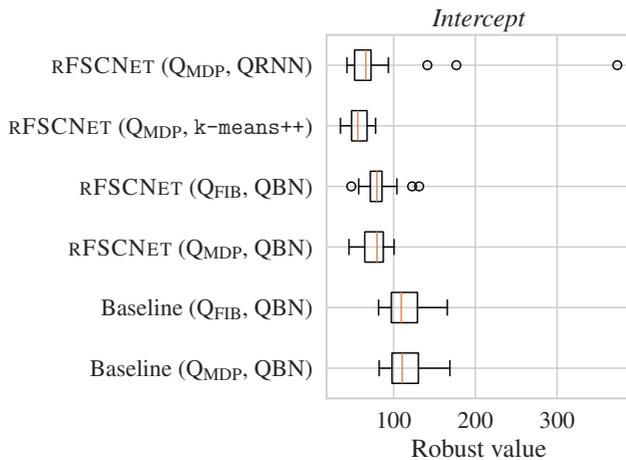


Figure 5. Boxplots of the best robust values across seeds on multiple configurations tested for RFSCNET on the Intercept RPOMDP and a baseline on a fixed (nominal) POMDP instance.

7.2 Discussion

The performance of the FSCs is impacted by the quality of discretization of the hidden states of the RNN, and faulty extraction of the FSC from the RNN leads to finding a robust value that is not informative and, consequently, a pessimistic POMDP that is not helpful. How to optimally extract finite-state representations of RNNs is still an open problem. In this paper, we tested multiple options based on clustering. We emphasize that the PIP framework is modular and allows other methods that compute FSCs for POMDPs to be used instead.

Finally, we note that the performance of RFSCNET is limited by the quality of the supervision policies we compute during training. This limitation could, for instance, explain why RFSCNET performs worst on *Intercept*, as this benchmark relies on information gathering, an aspect on which Q_{MDP} is known to perform poorly. Nonetheless, RFSCNET’s modularity allows for any POMDP policy to be applied as a supervision policy in the RNN’s training procedure, allowing for trade-offs between quality and computational efficiency.

8 Related Work

Early works on RPOMDPs extended value iteration and point-based methods to account for the additional uncertainty [29, 51], or use sampling over the uncertainty sets [10]. Nakao et al. [46] extend value iteration for *distributionally robust* POMDPs, where the agent receives side information after a decision period, which is a less conservative setting. Extensions to value iteration for RPOMDPs typically do not scale well to the large state spaces (up to 13000+) we consider in this paper. Ni and Liu [47, 48] introduce a policy iteration algorithm for optimistic policies, which does not extend to the robust setting we consider. Chamie and Mostafa [12] consider robustifying a given POMDP policy to combat sensor imprecision. Recent methods compute FSCs for finite sets of POMDPs through subgradients [20] and for RPOMDPs via quadratic [61] or sequential [16] convex programming, with the latter outperforming the former. In contrast to our work, the convex optimization methods compute FSCs of a pre-defined size and structure and cannot handle the optimistic case. RNNs have previously been used in a planning setting to compute FSCs for (non-robust) POMDPs [11]. Yet, robustness against model uncertainty was not considered, and, to the best of our knowledge,

no method exists that exploits the learning capabilities of RNNs in a robust planning setting.

9 Conclusion

In this paper, we presented PIP, a novel planning framework for RPOMDPs, and our algorithm, RFSCNET, that is based on the PIP framework. RFSCNET utilizes RNNs to compute FSCs for the pessimistic POMDPs selected by PIP, allowing the memory structure to be learned from data. Our experiments show that our approach yields more robust policies than the baseline approaches. Additionally, RFSCNET is less sensitive to over-parameterization of the memory size than SCP, a state-of-the-art solver. Furthermore, RFSCNET outperforms SCP in a subset of the benchmarks considered in this paper. Future work may investigate alleviating the limitation of the supervision policies by optimizing the RNN with a more sophisticated training objective or by considering more advanced supervision policies.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback, Eline M. Bovy and Merlijn Krales for helpful discussions on earlier versions of this work, and Niels Neerhoff for providing the code from his master’s thesis. This research has been partially funded by the NWO grant NWA.1160.18.238 (PrimaVera), the FWO grant GOAH524N (SynthEx), and the ERC Starting Grant 101077178 (DEUCE).

References

- [1] M. Abadi, P. Barham, J. Chen, Z. Chen, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- [2] D. Aberdeen. *Policy-gradient algorithms for partially observable Markov decision processes*. PhD thesis, The Australian National University, 2003.
- [3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035. SIAM, 2007.
- [4] K. J. Åström. Optimal control of Markov processes with incomplete state information. *J. Math. Anal. Appl.*, 10(1):174–205, 1965.
- [5] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [6] D. P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.
- [7] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, 1991.
- [8] B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *IJCAI*, pages 1641–1646, 2009.
- [9] E. M. Bovy, M. Suilen, S. Junges, and N. Jansen. Imprecise probabilities meet partial observability: Game semantics for robust POMDPs. In *IJCAI*, pages 6697–6706. ijcai.org, 2024.
- [10] B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *ICRA*, pages 3313–3318. IEEE, 2007.
- [11] S. Carr, N. Jansen, and U. Topcu. Task-aware verifiable RNN-based policies for partially observable Markov decision processes. *J. Artif. Intell. Res.*, 72:819–847, 2021.
- [12] M. E. Chamie and H. Mostafa. Robust action selection in partially observable Markov decision processes with model uncertainty. In *CDC*, pages 5586–5591. IEEE, 2018.
- [13] K. Chatterjee, M. Chmelik, R. Gupta, and A. Kanodia. Qualitative analysis of pomdps with temporal logic specifications for robotics applications. In *ICRA*, pages 325–330. IEEE, 2015.
- [14] T. Chen, T. Han, and M. Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.
- [15] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734. ACL, 2014.

- [16] M. Cubuktepe, N. Jansen, S. Junges, A. Marandi, M. Suilen, and U. Topcu. Robust finite-state controllers for uncertain POMDPs. In *AAAI*, pages 11792–11800. AAAI Press, 2021.
- [17] M. Fairbank and E. Alonso. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *IJCNN*, pages 1–8. IEEE, 2012.
- [18] V. Forejt, M. Z. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011.
- [19] M. F. L. Galesloot, M. Suilen, T. D. Simão, S. Carr, M. T. J. Spaan, U. Topcu, and N. Jansen. Pessimistic iterative planning with RNNs for robust POMDPs. *CoRR*, abs/2408.08770, 2024.
- [20] M. F. L. Galesloot, R. Andriushchenko, M. Češka, S. Junges, and N. Jansen. Robust finite-memory policy gradients for hidden-model POMDPs. In *IJCAI*, 2025. (to appear).
- [21] M. Grzes, P. Poupart, X. Yang, and J. Hoey. Energy efficient execution of POMDP policies. *IEEE Trans. Cybern.*, 45(11):2484–2497, 2015.
- [22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL <https://www.gurobi.com>.
- [23] E. A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *NIPS*, pages 1015–1021. The MIT Press, 1997.
- [24] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 13:33–94, 2000.
- [25] C. Hensel, S. Junges, J. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610, 2022.
- [26] T. J. Hickey, Q. Ju, and M. H. van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, 2001.
- [27] K. Horák, B. Bosanský, and K. Chatterjee. Goal-hsvi: Heuristic search value iteration for goal POMDPs. In *IJCAI*, pages 4764–4770. ijcai.org, 2018.
- [28] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, 2017.
- [29] H. Itoh and K. Nakamura. Partially observable Markov decision processes with imprecise parameters. *Artif. Intell.*, 171(8-9):453–490, 2007.
- [30] G. N. Iyengar. Robust dynamic programming. *Math. Oper. Res.*, 30(2):257–280, 2005.
- [31] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J. Katoen, and B. Becker. Finite-state controllers of POMDPs using parameter synthesis. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 519–529. AUAI Press, 2018.
- [32] S. Junges, N. Jansen, and S. A. Seshia. Enforcing almost-sure reachability in POMDPs. In *Computer Aided Verification (CAV)*, volume 12760 of *Lecture Notes in Computer Science*, pages 602–625. Springer, 2021.
- [33] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.
- [34] S. Kalyanasundaram, E. K. P. Chong, and N. B. Shroff. Markov decision processes with uncertain transition rates: sensitivity and robust control. In *CDC*, pages 3799–3804. IEEE, 2002.
- [35] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [36] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Ure, and J. Vian. *Optimized Airborne Collision Avoidance*, pages 249–276. MIT press, 2015.
- [37] A. Koul, A. Fern, and S. Greydanus. Learning finite state representations of recurrent policy networks. In *ICLR (Poster)*. OpenReview.net, 2019.
- [38] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*. MIT Press, 2008.
- [39] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.
- [40] G. Lambrechts, A. Bolland, and D. Ernst. Recurrent networks, hidden states and beliefs in partially observable environments. *Trans. Mach. Learn. Res.*, 2022, 2022.
- [41] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning Policies for Partially Observable Environments: Scaling Up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann, 1995.
- [42] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34, 2003.
- [43] N. Meuleau, K. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by searching the space of finite policies. In *UAI*, pages 417–426. Morgan Kaufmann, 1999.
- [44] T. M. Moerland, J. Broekens, and C. M. Jonker. Model-based reinforcement learning: A survey. *CoRR*, abs/2006.16712, 2020.
- [45] E. Muškardin, B. K. Aichernig, I. Pill, and M. Tappler. Learning finite state models from recurrent neural networks. In *Integrated Formal Methods*, pages 229–248. Springer International Publishing, 2022.
- [46] H. Nakao, R. Jiang, and S. Shen. Distributionally robust partially observable markov decision process with moment-based ambiguity. *SIAM J. Optim.*, 31(1):461–488, 2021.
- [47] Y. Ni and Z. Liu. Bounded-Parameter Partially Observable Markov Decision Processes. In *ICAPS*, pages 240–247. AAAI, 2008.
- [48] Y. Ni and Z. Liu. Policy iteration for bounded-parameter POMDPs. *Soft Comput.*, 17(4):537–548, 2013.
- [49] A. Nilim and L. E. Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, 2005.
- [50] C. W. Omlin and C. L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.
- [51] T. Osogami. Robust partially observable Markov decision process. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 106–115. JMLR.org, 2015.
- [52] S. D. Patek. *Stochastic and shortest path games: theory and algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1997.
- [53] S. D. Patek and D. P. Bertsekas. Stochastic Shortest Path Games. *SIAM Journal on Control and Optimization*, pages 804–824, 1999.
- [54] H. Peters. *Game theory: A Multi-leveled approach*. Springer, 2015.
- [55] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE, 2006.
- [56] P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*, pages 823–830. MIT Press, 2003.
- [57] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011.
- [58] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2164–2172. MIT Press, 2010.
- [59] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 520–527. AUAI Press, 2004.
- [60] M. T. J. Spaan. Partially Observable Markov Decision Processes. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, pages 387–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [61] M. Suilen, N. Jansen, M. Cubuktepe, and U. Topcu. Robust policy synthesis for uncertain POMDPs via convex optimization. In *IJCAI*, pages 4113–4120. ijcai.org, 2020.
- [62] M. Suilen, T. S. Badings, E. M. Bovy, D. Parker, and N. Jansen. Robust markov decision processes: A place where AI and formal methods meet. In *Principles of Verification (3)*, volume 15262 of *Lecture Notes in Computer Science*, pages 126–154. Springer, 2024.
- [63] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.
- [64] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pages 23–30. IEEE, 2017.
- [65] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proc. IEEE*, 78(10):1550–1560, 1990.
- [66] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *ICANN (1)*, volume 4668 of *Lecture Notes in Computer Science*, pages 697–706. Springer, 2007.
- [67] W. Wiesemann, D. Kuhn, and B. Rustem. Robust Markov decision processes. *Math. Oper. Res.*, 38(1):153–183, 2013.
- [68] Z. Zeng, R. M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Comput.*, 5(6):976–990, 1993.
- [69] W. Zheng, T. Jung, and H. Lin. The stackelberg equilibrium for one-sided zero-sum partially observable stochastic games. *Autom.*, 140:110231, 2022.