Conviction-Based Planning for Sparse Reward Reinforcement Learning Problems

Simon Ouellette, Éric Beaudry, Mohamed Bouguessa

Université du Québec à Montréal

Abstract

Deep reinforcement learning (RL) methods require a large amount of interactions, making them difficult to use in real-world applications. This is especially true when the rewards are sparse since random or semi-random exploration struggles to find them. Learning from Demonstrations (LfD) mitigates this by eliminating the need for random exploration. So far, most LfD solutions have been based on model-free RL approaches that struggle with tasks that require planning. We propose a new algorithm that successfully combines model-based RL and LfD by leveraging the notion of uncertainty in the transition model during planning. We also introduce the concept of conviction, an uncertainty-to-reward ratio used to decide on optimal actions while planning. Our approach significantly outperforms the relevant baselines in the Minigrid and Sokoban environments.

Introduction

Deep reinforcement learning has been very successful in certain areas (Schrittwieser et al. 2020a; Vinyals et al. 2019; Berner et al. 2019; Badia et al. 2020) where it is possible to run countless simulations and thus learn from an unlimited number of interactions. However, for robotics, one of its main limitations is precisely the need for a large number of interactions. It also struggles when the reward is sparse since it is difficult for random exploration to reach that reward within a reasonable number of interactions. This makes deep reinforcement learning prohibitively slow for most real-world problems (and sometimes unsafe when dealing with dangerous or fragile robotic equipment) (Deisenroth, Fox, and Rasmussen 2013; Pignat and Calinon 2019; Deisenroth and Rasmussen 2011) .

Various categories of methods exist to mitigate the sparse reward problem in reinforcement learning (RL). One such category is the idea of "reward shaping". It consists of applying hand-crafted modifications to the reward function, such as adding intermediate rewards for reaching intermediate goals. Another category encompasses more sophisticated exploration strategies (e.g., curiosity-driven exploration) (Ladosz et al. 2022). A special case of that consists of "Learning from Demonstrations" (LfD), in which the models are trained from expert demonstrations of solutions rather than model-driven exploration.

This paper presents an approach that falls under the LfD category, although we will resort to a bit of reward shaping for our experiments on Minigrid (Chevalier-Boisvert, Willems, and Pal 2018). To the best of our knowledge, existing LfD approaches rely exclusively on model-free paradigms. Although model-free RL approaches can be very effective when there is a sufficient number of data samples, it can be difficult to learn policies that generalize well in highly combinatorial environments (e.g., planning intensive) (Racanière et al. 2017; Schrittwieser et al. 2020b; Moerland, Broekens, and Jonker 2020).

However, combining model-based RL with LfD is not trivial. This is because human expert demonstrations are not complete explorations of the full transition space. Instead, they typically represent the (quasi-) optimal subset of possible state transitions (minus the accidental errors). Therefore, it is difficult for model-based algorithms to learn an unbiased and complete version of the environment dynamics from only a demonstration dataset. As a result, attempting to predict trajectories from state transitions that were either never observed or rarely observed leads to arbitrary extrapolated values, thereby derailing planning.

To address the abovementioned issue, we present a novel strategy for combining the advantages of LfD and model-based RL. We achieve this by elaborating a planning algorithm that leverages uncertainty in the learned transition model to pre-prune planning trajectories. Additionally, our algorithm searches for the sequence of actions that maximizes *conviction*, which we define as the ratio of expected value to cumulative prediction uncertainty. We conducted detailed experiments on the Minigrid (Chevalier-Boisvert, Willems, and Pal 2018) and Sokoban (Schrader 2018) environments, and demonstrated that it significantly outperforms all relevant baselines in low data regimes.

The significance of this work can be summarized as follows:

1. We propose a novel strategy for sparse reward RL problems that outperforms state-of-the-art approaches on Sokoban, in terms of sample efficiency.
2. We introduce the concept of conviction, which is used in the aforementioned strategy.
3. We devise variants of Iterative Deepening A* (IDA*) and Monte-Carlo Tree Search that leverage the notions of conviction and uncertainty to find solutions from an

incomplete world model.

Related work

Dealing with sparse rewards in RL

Some reinforcement learning problems only provide a reward after a long sequence of specific actions that lead to the desired goal(Ladosz et al. 2022). This makes the traditional reinforcement learning approach of random exploration quite difficult to apply, as it is unlikely that a sequence of random actions will reach this elusive reward.

Various paradigms have been proposed to address this sparse reward challenge. One such approach, *reward shaping*, consists of artificially imposing a more suitable reward function on top of the intrinsic one. This hand-crafted reward function generally provides intermediate goals that are easier to attain while being a general indicator of progress toward the goal (Hu et al. 2020; Trott et al. 2019; Eschmann 2021).

Another paradigm uses different exploration algorithms than purely random actions. An example is curiosity-driven exploration (Ladosz et al. 2022; Eschmann 2021; Pathak et al. 2017). In this method, the model attempts to predict the next state of the environment. Prediction errors are treated as a positive reward related to surprise. The agent will then attempt to visit these states of surprise more often since they are states for which the model has incomplete information. Part of this paradigm is the method of Learning from Demonstrations (LfD). Since this is the main focus of this work, we discuss it in more detail in the following section.

Learning from Demonstrations

In LfD, the policy is learned at least partially through the use of expert demonstrations rather than random exploration alone. The motivation is twofold: (1) to reduce the number of interactions with the environment as much as possible and (2) to work around reward sparsity by offering complete examples of solutions (and, therefore, rewards) to the model.

LfD should not be confused with Inverse RL. They have significant overlap in their techniques since they both rely on expert demonstrations, but their goals are different. In Inverse RL, the goal is to infer the reward function from user examples. In other words, the Inverse RL literature does not express concern for sparse rewards or sample efficiency. Instead, it is motivated by the fact that for some problems, it is difficult to explicitly describe the reward function because it is too complex, elusive, or subtle (Abbeel and Ng 2004).

Deep Q-Learning from Demonstrations (DQfD) is a hybrid model-free RL algorithm that combines exploration and LfD (Hester et al. 2018). It can reach state-of-the-art performance on tasks from the Arcade Learning Environment from much fewer samples. In addition, due to its ability to learn from exploration, it can exceed the demonstration performance on a significant number of cases. They compared DQfD to the Prioritized Dueling Double Deep Q-Network(Schaul et al. 2015) algorithm, which does not use demonstrations. In the games *Hero*, *Pitfall*, and *Road Runner*, the human demonstrations enable DQfD to achieve better performance than any previously published result, especially from the standpoint of sample efficiency.

DQfD achieves this via a special supervised learning loss term in which, for a given state in a demonstration, the unobserved actions' Q-values are penalized. Soft Q Imitation Learning (SQIL)(Reddy, Dragan, and Levine 2019), Normalized Actor-Critic (Gao et al. 2018), Monte Carlo augmented Actor-Critic (Wilcox et al. 2022), Cycle-of-Learning (Goecks et al. 2020) and Deep Deterministic Policy Gradient from Demonstrations (DDPGfD) (Vecerik et al. 2017) all use the same core idea, along with pretraining on expert demonstrations. SQIL does not require an explicit reward from the environment, while Monte-Carlo augmented Actor-Critic uses actor-critic instead of Q-Learning.

Implicit planning

There is a line of research that involves using model-free approaches with neural architectures that are structured to learn to plan implicitly. The Deep Repeated ConvLSTM (Guez et al. 2019) is a recurrent algorithm that allows iterations over a latent state maintained by stacked ConvLSTM layers. This architecture can learn to predict several moves ahead, since each iteration of the loop can model an interaction in its latent state.

Working Memory Graphs (Loynd et al. 2020) (WMG) are the state of the art in that line of research. It uses a Transformer architecture along with preprocessed, factored observations. They outperform previous implicit planning approaches on Sokoban, such as Deep Repeated ConvLSTMs. This approach will be used as a reference algorithm in our experiments on Sokoban.

Methodology

Problem formulation

The theoretical framework for reinforcement learning relies on the Markov Decision Process formulation for sequential decision making (Sutton and Barto 2018). It is defined as a tuple (S, A, R, P, γ), where:

- S is a set of discrete states;
- A is a set of discrete actions;
- $R(s, a)$ is a mapping from state-action to reward;
- $P(s_{t+1}|s_t, a)$ is a mapping from state-action to the next state;
- γ is a discount factor, $0 < \gamma \leq 1$.

At each observed state $s_t \in S$, the agent takes an action $a \in A$. In the subsequent step, the agent receives a reward $R(s, a)$ and observes a new state s_{t+1} based on $P(s_{t+1}|s_t, a)$. In the sparse rewards case of interest to us, in the vast majority of cases $R(s, a)$ returns a zero (or negative) reward. The goal of the agent is to learn a behavior policy π that will maximize the expected discounted total reward. Markov Decision Processes have the Markov property, which is that the current state s_t contains all of the necessary information to predict the next state s_{t+1}. Formally:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, S_2, \ldots, S_t) \tag{1}$$

In LfD, instead of observing a direct exploratory interaction with the environment, we have an expert who demonstrates successful interactions with the environment. The collected data is then fed to the model for training. In hybrid solutions like DQfD, this can be a pre-training phase, subsequently followed by learning from exploration in the environment.

Model-based RL

Description In model-based RL, we have a model of the environment dynamics:

$$D : (s_t, a_i) \to \hat{s}_{t+1} \tag{2}$$

That is, given a current state s_t and an action a_i, it produces a prediction $\hat{s}_{t+1}$ of the upcoming state. We also have an action-value model:

$$Q^\pi : (s_t, a_i) \to E\Big[\sum_{k=0}^{K} \gamma^k r_{t+k}\Big] \tag{3}$$

This is the discounted sum of rewards (over the remaining K interactions) that are expected to be obtained by applying action a_i in state s_t if we subsequently use the policy π. This policy consists of a planning algorithm (which can be learned) that searches for the sequence of actions to maximize the total cumulative reward given the transition model D and the value model Q^π.

Advantage Given a sufficiently combinatorially complex environment, it may be considered implausible (and undesirable) to observe demonstrations over all possible sequences of events, or even to cover that space sufficiently to expect robust interpolation by a neural network. An agent that lacks the ability to plan will need to extrapolate, and thus underperform in sequences of states that are far from the observed ones. Model-based RL moves away from merely interpolating over previously seen trajectories towards a more fundamental estimation of environment dynamics coupled with planning ahead. This tends to generalize better than directly learning mappings from state to action as in model-free RL (Racanière et al. 2017; Schrittwieser et al. 2020b; Moerland, Broekens, and Jonker 2020).

Disadvantage In the sparse reward context, random exploration presents challenges in observing positive rewards, making it difficult to learn the action-value function. The probability of reaching the rare reward for a randomly generated sequence of actions is very low, which means that a much higher number of such interaction sequences must be made in order to get positive reward examples. Because the action-value model needs to observe a significant number of such positive rewards to learn anything, this sets a much higher lower bound on the number of interactions required (compared to an LfD approach).

Learning from Demonstrations

Description Each demonstration step is a tuple (s, a, s', r) consisting of a preliminary state s, an action taken a, a resulting state s', and reward r. In the case of hybrid solutions, the policy is pre-trained on these demonstration sequences, and a second phase of exploration-driven RL training is made. In the case of our proposed approach, the transition model and the value model are trained entirely from these demonstrations, and no exploration-driven RL training is needed.

Algorithm	**Sparse Rewards**	**Ability to plan**
Model-based RL	No	Yes
Model-free LfD	Yes	No
Conviction-based Planner	Yes	Yes

Table 1: Comparison of Model-based RL, LfD, and our Conviction-based Planner approach.

Advantage Because the expert demonstrations all include rare rewards, this allows to train the value model from relatively few interactions, compared to random exploration. Expert demonstrations can be thought of as much more reward-dense interaction sequences than those obtained from random exploration.

Disadvantage Model-free LfD is not equipped with a planning algorithm or a model of the environment (by definition). Therefore, its direct policy must learn to map states to actions. This is difficult when the environment presents a combinatorial explosion of possible state-action sequences.

Combining Model-based RL and LfD

We seek to combine Model-based RL and LfD to obtain the sample-efficiency advantages of both (see Table 1). Combining model-based RL with LfD suggests that the transition model is learned from expert demonstrations rather than (only) from random exploration. Expert demonstrations are sequences of (quasi-)optimal actions that lead to solutions for each provided problem. Most of the time, these state-action transitions do not cover the whole range of possible state-action transitions because invalid, nonsensical, or counter-productive state-action transitions are not usually attempted in expert demonstrations.

For example, in Minigrid[1], a human expert will not voluntarily try to move into a wall or move into a closed door. Because of this, a transition model trained from expert demonstrations is generally incomplete. If one were to query it for state-action transitions that were never (or very rarely) observed during training, it would give unreliable predictions. In Minigrid, for instance, we found that the transition model would predict that the agent will move through the wall or through the closed door if we asked it what happens when we move forward in such a state.

This is a problem for model-based RL because the planning algorithm, whether it is IDA*, Monte-Carlo Tree

[1]Minigrid is a grid-based environment in which an agent can navigate rooms and open doors, while trying to reach a specific goal cell. A detailed description of Minigrid is provided in the Experiments section.

Search or other, will necessarily query such invalid state-action transitions. The predicted outcome may be favorable, even though it is incorrect, which will derail planning. This is why naively combining Model-based RL, and LfD is relatively ineffective, as will be shown empirically in our ablation studies (*NaivePlanner-MCTS* and *NaivePlanner-IDA**).

The Proposed Strategy

The strategy that we propose in order to leverage the benefits of model-based RL and LfD is to get an estimate of uncertainty in the transition model and use it to ignore highly uncertain predictions during planning. The key elements of our proposed strategy consist of:

- An encoder that shapes the encoding space in a way that is semantically meaningful.
- A mechanism to return prediction uncertainty in the transition model.
- A modification to the planning algorithm that prunes planning paths whose uncertainty exceeds a threshold while prioritizing high value-to-uncertainty ratios (that we call conviction).

The mechanism that returns prediction uncertainty in the transition model can be implemented in a number of ways, such as Bayesian approaches or Deep Neural Network Ensembles (Lakshminarayanan, Pritzel, and Blundell 2017). The method we used technically fits neither category but can be interpreted as an implementation of a Gaussian Process. We save the latent space encodings from the training dataset in an in-memory table. At inference time, we essentially use k-nearest neighbors to retrieve the Euclidean distance between the generated encodings and the nearest encoding seen during training. This distance is treated as a proxy for uncertainty in our model: if the distance is zero, it means we have already seen this state during training; therefore, one can expect the prediction to be trustworthy.

The success of this uncertainty-based approach depends on whether the latent space encodings are semantically meaningful. What is meant by that is that there should be no unnecessary distinction between semantically equivalent states, yet there should be sufficient distinctions to separate the semantically different states. Unnecessary distinctions between semantically equivalent states imply an unnecessarily inflated encoding space. That is, the dimensionality of the encoding space needs to be larger in order to be able to represent the additional, superfluous distinctions. Doing so, however, reduces the sample efficiency of the algorithm because more examples are needed to obtain a dataset that properly covers that larger encoding space. On the other hand, the lack of necessary semantic distinctions will result in incorrect transition predictions.

To give a concrete example of the impact of semantically meaningful distinctions in the encoder, we can look at Minigrid. In this problem domain, some levels have doors of different colors. However, the color of the door has no impact on the solution. In order to get a compact and efficient latent space, the encodings should be such that doors of different colors in the raw observation space are not encoded as distinct objects in the encoding space. In contrast, the state of being in front of a closed door and the state of being in front of an open door should be encoded as distinct states. The reason for this is that the state of a door is crucial for the solution: the agent cannot pass through it without first opening the door.

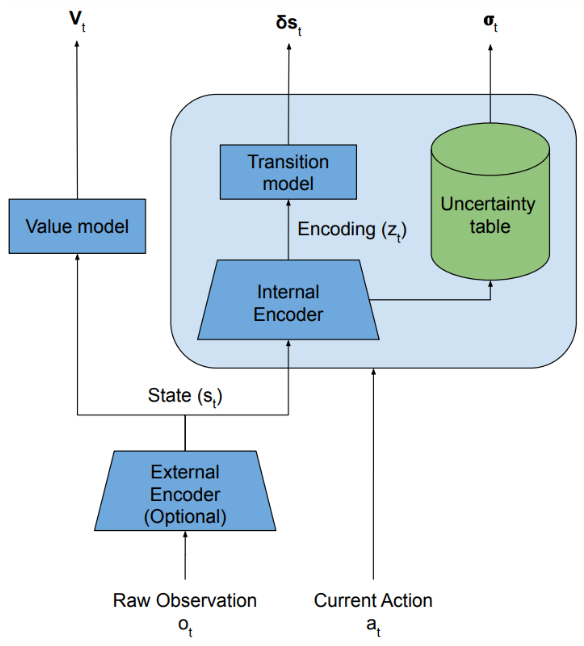

Figure 1: Architecture of the Conviction Planner.

Learning an encoder that generates semantically meaningful solutions does not necessarily happen on its own, which means that it is generally not possible to use a trained encoder. Furthermore, it is not possible to train an encoder in a supervised way for this purpose, since the encoder itself would be subject to the uncertainty present in the training set. Instead, this encoder must be a hard-coded module that takes as input a state and action and outputs the most compressed, semantically meaningful encoding. This feature engineering effort is, in a sense, comparable to the type of effort required in reward shaping to make a sparse reward problem learnable by traditional RL methods. There is “no free lunch” in machine learning, after all, and if we want to gain sample efficiency, we must compensate with strong priors.

The Model Architecture

The model architecture closely follows typical Model-based RL. As depicted in Figure 1, our approach consists of three main modules:

- the internal encoder: $E : (s_t, a_t) \rightarrow z_t$.
- the value model: $V : s_t \rightarrow \hat{V}_t$.
- the transition model: $T : (z_t, a) \rightarrow (\delta s | a, \sigma_t)$.

For Minigrid, the value model simply corresponds to the reward associated with state s_t (0 everywhere except when reaching the goal). For Sokoban, the details vary (see subsection ”Sokoban” of the Experiments section).

In the transition model, $\delta s_t | a$ is the predicted differential of the input vector s_t as a result of the application of the action a. σ_t is the distance of z_t to the nearest encoding in the uncertainty table, which is used as proxy for the notion of uncertainty. The transition model acts like a residual connection: its output is a predicted difference between the current and the next frame representation, which is added onto the input.

The optional external encoder would be necessary in cases where the environment does not output a symbolic observation. In such a case, it would be helpful to have a Convolutional Neural Network (e.g., an autoencoder) that condenses the raw observation into a manageable state space. This state space is where the planning iterations occur. In our experiments, we deal with symbolic representations, so $o_t = s_t$.

The loss function that is minimized by the Conviction-based Planner during training is:

$$L = L_s + L_V \tag{4}$$

, where:

$$L_s = \sum (\delta \hat{s}_{t+1} | a'_t - \delta s_{t+1} | a'_t)^2 \tag{5}$$

$$L_V = \sum (\hat{V}_t - V_t)^2 \tag{6}$$

In the loss function, $\delta \hat{s}_{t+1} | a'_t$ is the predicted change in state from applying the demonstration action a'_t, while $\delta s_{t+1} | a'_t$ is the corresponding ground truth. Note that the uncertainty output σ_t of the transition model is not part of the loss function, it is trained in an unsupervised way. It consists of the nearest neighbor's Euclidean distance in the encoding space.

At each epoch of the training procedure, we sample a batch from the demonstrations. Optionally, the raw observation is passed through the external (e.g., convolutional) encoder. The state s_t is fed to the internal encoder E, which outputs z_t. This is then fed to the transition model T, which outputs its predicted state delta δs_t. Concurrently, s_t is fed directly to the value model, which outputs its value prediction $\hat{V}_t$. We then have all of the necessary predictions to calculate the loss function and apply gradients.

Conviction-Based Planning

Once the transition and value models have been trained, we can start solving tasks with the help of a planning algorithm. In our experiments, we use two algorithms: Iterative Deepening A* (with a learned heuristic function) (Korf 1985) and Monte Carlo Tree Search (Coulom 2006). In both cases, the general idea is that we start from a given state s_t of the environment. The algorithm then iteratively searches for the sequence of actions that leads to a final predicted state $\hat{s}_{t+K}$ for which the value model returns the best value. This search is made possible by using the transition model to predict the state of the environment after each suggested action.

A notable feature of our proposed solution is that we leverage information about uncertainty while planning, since our transition model cannot be trusted for any arbitrarily suggested state-action transition. Instead of merely optimizing for the predicted value of the end state, we optimize a metric ψ, which we call conviction, that is mathematically described as follows:

$$\psi = \frac{\hat{V}_{t+K}}{\sum_{\tau=t}^{t+K} \sigma_\tau + \epsilon} \tag{7}$$

In Eq. (7), the denominator consists of the cumulative uncertainty of actions in the planned sequence of length K, ϵ is a very small constant to prevent division by zero.

Additionally, as a search optimization, we prune sub-trees when a particular transition prediction is associated with an uncertainty that surpasses a given threshold (which is an empirically determined hyper-parameter).

Experiments

Since our proposed solution combines model-based RL and LfD approaches, we compared those two strategies individually. Models *ModelBasedRL-IDA** and *ModelBasedRL-MCTS* implement traditional model-based RL with the same planning algorithms (except for the conviction-related modifications) as our proposed solutions *ConvictionPlanner-IDA** and *ConvictionPlanner-MCTS* respectively.

For our LfD benchmark, we use Deep Q-Learning from Demonstrations (DQfD) (Hester et al. 2018) because it is the state of the art in discrete environments. We run three different experiments from this algorithm: *DQfD-PreTraining*, *DQfD-Hybrid* and *DQfD-Model-Free*.

DQfD-PreTraining consists of only the pre-training phase from demonstrations. We do not perform any random exploration phase, and as such the policy is learned strictly from supervised training on expert demonstrations. It is essentially equivalent to *Behavioral Cloning* methods. *DQfD-Model-Free* uses the DQfD algorithm without any pre-training on demonstrations, so it is essentially the same as standard Deep Q-Learning. *DQfD-Hybrid* implements both aspects of DQfD.

We consider what happens if we train a Model-based RL approach on a mixture of demonstrations and random exploration. Presumably, this would eliminate the need to handle uncertainty, since the random explorations would cover the transition space. This is the experiment *HybridPlanner-IDA**, which we carry out on the Sokoban environment.

We also perform an ablation study to show that uncertainty-aware planning is necessary to combine Model-based RL and LfD in an effective way. *NaivePlanner-IDA** and *NaivePlanner-MCTS* are exactly the same models as *ConvictionPlanner-IDA** and *ConvictionPlanner-MCTS*, with the notable exception that planning optimizes estimated value only (rather than conviction) and it does not prune sub-trees based on uncertainty.

Finally, we compare to Working Memory Graphs (WMG), using the unfiltered Boxoban levels as in (Loynd et al. 2020), instead of our usual 3-box randomized levels. See Table 5 for results. Due to the lower sample efficiency of WMG, we must compare our highest number of interactions on the *ConvictionPlanner* with WMG results at much higher numbers of interactions. The number of demonstrations required to test at these higher data regimes is prohibitive, which is why we stop at 50K for our algorithm. It is, after all, intended for lower data regimes.

The evaluation procedure involves training instances of the compared models on different amounts of interaction steps. We then assess each of these models on 100 test problem instances. The test problem instances were generated after the training phase, with unique random seed values. These problem instances have never been observed by the agents during their training phase, but they have the same grid dimensions. The success rate is the percentage of problem instances, out of the 100, solved by the given model (only 1 trial per problem instance). From these results, we get a portrayal of the sampling efficiency of each method.

Minigrid

Minigrid(Chevalier-Boisvert, Willems, and Pal 2018) is a simple grid-based world that an agent must navigate, using a small set of possible actions. The goal is typically to reach an end-goal cell, although Minigrid contains many types of levels, some of which have different goals (such as picking up certain objects). The levels used in this work will all have the same goal: to reach a specific goal cell (visually represented as a green cell).

This environment is partially observable. The agent only ever observes a 7×7 window in front of itself. To address the partial observability, we use an in-memory grid that remembers the revealed cells. This allows us, during planning, to determine if trajectories reveal new cells, which is encouraged via an intermediate reward. In addition to maintaining a global state of the grid (with revealed and hidden cells), this module also remembers its absolute position in that grid, as well as the direction in which it is facing. This is also necessary for the planning procedure.

We use two different types of levels:

1. *MiniGrid-FourRooms-v0*: which uses a 19×19 grid (see Appendix A). There are three possible actions: forward, turn right, and turn left.
2. *MiniGrid-MultiRoom-N6-v0*: which uses a 25×25 grid (see Appendix A). There are four possible actions: forward, turn right, turn left, and toggle door open/closed.

The agent receives a positive reward when the end goal is reached (the green cell); otherwise, it gets a zero reward. During planning, an intermediate reward is used to encourage exploration: reaching a previously unexplored cell adds +1 to the cumulative reward, while reaching the end goal cell adds +25 to the reward. As a result, the planning algorithm will select and execute the action sequence that maximizes the number of revealed cells within the K planned steps. If the goal cell is already revealed, it will tend to favor reaching this goal cell due to its higher reward.

According to the documentation on Minigrid, the *MiniGrid-MultiRoom-N6-v0* level is difficult to solve using standard RL techniques. This is mainly due to the sparse reward. A long sequence of optimal actions must be executed to reach that sparse reward. By the same logic, the *MiniGrid-FourRooms-v0* should also be quite difficult (though, perhaps, to a lesser extent). Because Iterative Deepening A* is not appropriate in a partially observable environment (the end goal is not known unless revealed), our planning algorithm will be Monte-Carlo Tree Search.

Conviction-based MCTS

Here we describe the changes that we made to the regular Monte-Carlo Tree Search algorithm. For the sake of brevity we assume that the reader is already familiar with Monte-Carlo Tree Search.

In the node expansion and simulation functions of MCTS, a function is usually called in order predict the next state s_{t+1} as a function of the current s_t state and a proposed action a. This function uses the transition model and the value models to make state transition and reward predictions. In our implementation, this function also returns a notion of uncertainty associated with that prediction.

Furthermore, it includes some logic relating to the ”intermediate reward” that encourages exploration. This change is specific to Minigrid, not to our conviction-based approach in general. Whenever the predicted transition lands the agent on a previously unexplored cell, we set the reward to 1.

In the node expansion function, the uncertainty returned by the transition prediction function is used to determine whether we can continue expanding the tree in this direction or not. If the uncertainty is too high for the proposed action from the given node, we truncate the entire sub-tree starting from that edge.

In the simulation function, we use this uncertainty in two ways. First, we ignore a transition prediction in the simulation roll-out loop if its uncertainty is too high. That is, it does not count as part of the final value estimate or state transition sequence. Furthermore, we use it to calculate the conviction metric. This is the value that is backpropagated and optimized for, rather than what we usually refer to as the Q-value.

Sokoban

Sokoban(Schrader 2018) is also based on a grid world. The goal is to push boxes onto special cells identified as goals. There is an equal number of boxes and goal cells, and once all of the former have been pushed onto the latter, the level is considered complete. The implementation of Sokoban that we use has four possible actions: up, down, left, and right. If the agent moves in a certain direction and there is a box there, it will push the box by one cell. If, however, there is a wall or another box behind that box, it will not move.

The agent receives a reward of +1 when it pushes a box onto a goal cell, -1 when it pushes a box that was on a goal cell to an adjacent non-goal cell, and +10 when it completes the level. The ConvictionPlanner approach will use our variant of IDA* as its planning algorithm.

While Minigrid is a good starting point to analyze our proposed algorithm, it is not as planning-intensive as Sokoban, nor is it as difficult to solve using LfD. This problem domain will allow us to truly differentiate the Conviction-based Planner from its LfD alternatives. Like Minigrid, the rewards are fairly rare, especially the final reward, and it is very difficult for random exploration to solve a level. Also, a great deal of planning has to be done to solve such a level, and potential deadlock situations have to be avoided (see Appendix B).

Conviction-based IDA*

Here, we describe the changes that we made to the regular IDA* algorithm. For the sake of brevity we assume that the reader is already familiar with IDA* (Korf 1985). In the regular IDA* algorithm, when we expand a node to obtain its children, a transition model is used to predict the new state s_{t+1} that will be seen from applying action a to state s_t. In our case, this function also returns an uncertainty value. For each potential child node, we only add it to the expanded tree if its associated uncertainty is below the provided threshold. This is the uncertainty-based pruning element of the algorithm. The conviction metric is used to prioritize the order in which child nodes are explored. Thus, the algorithm prioritizes higher-conviction trajectories.

In our IDA* implementation, instead of predicting the expected reward associated with a (s, a) transition, the value model predicts how many steps are left in the demonstration given s. We then plan with the goal of achieving a prediction of 0 steps left. To use the conviction formula we simply subtract V_t from an arbitrary large number, to get a value that increases as we approach the goal.

Results

We find that the ConvictionPlanner algorithm significantly outperforms all benchmark algorithms. On the *MiniGrid-FourRooms-v0* environment (see Table 2), *ConvictionPlanner-MCTS* outperforms baseline algorithms, except for *ModelBasedRL-MCTS*, that it only outperforms at 30 training interactions. For the *MiniGrid-MultiRoom-N6-v0* environment, it outperforms baselines with the exception of *DQfD-PreTraining*, which matches its performance at 100% when presented with 6,000 or more training samples (see see Table 3). On the Sokoban environment (see Table 4), *ConvictionPlanner-IDA** significantly outperforms reference algorithms at 65% (which was obtained after 30,000 training iterations). The second best is the *HybridPlanner-IDA**, at a corresponding accuracy of 38%.

Discussion

The importance of demonstrations

DQfD-Model-Free is unable to solve any of the problems. On all sample sizes tested, random exploration seldom made it out of the 2nd room in *MiniGrid-MultiRoom-N6-v0*. It never reached the end, so it never saw a non-zero reward. We see the same issue with model-based RL on Sokoban. The only reason why model-based RL performs so well on Minigrid is because of the intermediate rewards that encourage exploration in the planning algorithm.

The importance of planning

We hypothesize that, on Sokoban, it is tough from a given frame to predict the next-best action without reasoning through the steps. In other words, given two otherwise identical grids, a single difference in one cell can be enough to completely change the solution. This is probably why, on Sokoban, the second-best model is also an algorithm that uses explicit planning: *HybridPlanner-IDA**. On *MiniGrid-MultiRoom-N6-v0*, however, we hypothesize that a demonstration-based supervised policy performs relatively well because all the agent has to learn is that it needs to move towards the closed door that is visible in its partial view and then to open it.

In *MiniGrid-FourRooms-v0*, a lot of the frames will be "blank frames" that contain no obvious landmark to move towards. That is because the rooms tend to be much larger than in *MiniGrid-MultiRoom-N6-v0*. Therefore, we believe that this environment requires a bit more planning than *MiniGrid-MultiRoom-N6-v0*, though not as much as Sokoban. This can explain why *ModelBasedRL-MCTS* takes over *DQfD-PreTraining* as the second best approach in *MiniGrid-FourRooms-v0*.

Search tree pruning

The lower performance of *HybridPlanner-IDA** relative to *ConvictionPlanner-IDA** further motivates our conviction-based approach. This difference in performance is explained by the fact that the conviction-based approaches include subtree pruning based on uncertainty, which greatly reduces the search space. The hybrid planning approach does not do this, resulting in a larger search space and a worse performance. Thus, a functionality that was initially intended to only prevent invalid predictions in an incomplete model also became a way to avoid exploring sub-trees whose root node transition is *a priori* questionable, since it was never seen during training.

Limitations and assumptions

Our approach is more appropriate for problems where acquiring demonstrations and implementing the appropriate feature engineering in the encoder module is considered cheaper than running an indefinite number of exploration interactions. This is almost never the case for simulated environments or synthetic data. However, it can often be the case in robotics (physical environments) and expensive or otherwise limited datasets. Additionally, in this implementation we only handle discrete state spaces and actions. However, we believe that with minor modifications our approach can be generalized to continuous action spaces and states.

Comparison to WMG

WMG is intended to operate at a higher data regime than our algorithm. It is approximately 100 times less sample efficient than *ConvictionPlanner-IDA** on Sokoban, as shown by the fact that the 49% success rate is attained after 5M interactions for WMG versus 50K for *ConvictionPlanner-IDA** (see Table 5). Much like our approach, WMG requires problem-specific feature engineering. However, it is possible that in higher data regimes it outperforms *ConvictionPlanner-IDA**, since the feed-forward neural network used in our current implementation of IDA* is not as

Algorithm	Number of training steps							
	30	**750**	**1.5k**	**6K**	**9K**	**15K**	**22.5k**	**30k**
DQfD-PreTraining	1%	26%	35%	55%	55%	55%	57%	61%
DQfD-Hybrid	0%	16%	29%	65%	65%	67%	67%	68%
DQfD-Model-Free	0%	0%	0%	0%	0%	0%	0%	0%
ModelBasedRL-MCTS	5%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
NaivePlanner-MCTS	17%	20%	30%	32%	32%	34%	34%	34%
ConvictionPlanner-MCTS	**60%**	99%	99%	99%	**100%**	**100%**	**100%**	**100%**

Table 2: Success rate on 100 test examples, per training set size, on *MiniGrid-FourRooms-v0*.

Algorithm	Number of training steps					
	30	**750**	**1.5k**	**6K**	**9K**	**15K**
DQfD-PreTraining	0%	89%	96%	**100%**	**100%**	**100%**
DQfD-Hybrid	0%	35%	72%	95%	87%	94%
DQfD-Model-Free	0%	0%	0%	0%	0%	0%
ModelBasedRL-MCTS	28%	67%	64%	64%	64%	64%
NaivePlanner-MCTS	17%	20%	30%	32%	32%	34%
ConvictionPlanner-MCTS	**99%**	**99%**	**99%**	**100%**	**100%**	**100%**

Table 3: Success rate on 100 test examples, per training set size, on *MiniGrid-MultiRoom-N6-v0*.

Algorithm	Number of training steps						
	1K	**1.5k**	**5K**	**10K**	**15K**	**25K**	**30k**
DQfD-PreTraining	0%	0%	0%	0%	0%	0%	0%
DQfD-Hybrid	0%	0%	0%	0%	0%	0%	0%
DQfD-Model-Free	0%	0%	0%	0%	0%	0%	0%
WMG	0%	0%	0%	0%	0%	0%	0%
*ModelBasedRL-IDA**	0%	0%	0%	0%	5%	5%	6%
*NaivePlanner-IDA**	0%	0%	0%	1%	1%	2%	2%
*HybridPlanner-IDA**	1%	7%	26%	28%	31%	34%	38%
ConvictionPlanner-IDA*	**9%**	**12%**	**40%**	**58%**	**61%**	**64%**	**65%**

Table 4: Success rate on 100 test examples, per training set size, on Sokoban.

Algorithm	Number of training steps					
	30k	**50k**	**1M**	**2.5M**	**4M**	**5M**
WMG	0%	<1%	10.4%	33%	43%	49%
ConvictionPlanner-IDA*	**43%**	**49%**	N/A	N/A	N/A	N/A

Table 5: Success rate on 100 test examples, per training set size, on *Boxoban*.

powerful as a Transformer. The latter may be capable of learning search heuristics that are more efficient than the former.

Conclusion

We proposed a novel strategy to address the problem of learning inefficiencies caused by sparse rewards in reinforcement learning problems. It consists of combining an uncertainty-aware transition model, trained on demonstrations, with a variant of a search algorithm such as IDA* or MCTS. This variant uses uncertainty to prune subtrees during planning and to optimize trajectory selection based on the notion of conviction (the ratio of expected value to cumulative uncertainty), which we introduced. We tested it on deterministic discrete environments, namely Minigrid and Sokoban, in which it outperformed relevant ML baselines. In particular, it outperformed Working Memory Graphs on Sokoban in terms of sample efficiency, by a factor of approximately 100 times.

Possible future avenues of research include relaxing the assumption of discrete action and state spaces to support continuous environments. It would also be helpful to generalize the approach to stochastic environments. Finally, more work is needed to adapt this approach to non-symbolic problem domains.

References

Abbeel, P.; and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z. D.; and Blundell, C. 2020. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, 507–517. PMLR.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, 72–83. Springer.

Deisenroth, M.; and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 465–472. Citeseer.

Deisenroth, M. P.; Fox, D.; and Rasmussen, C. E. 2013. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2): 408–423.

Eschmann, J. 2021. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, 25–33.

Gao, Y.; Xu, H.; Lin, J.; Yu, F.; Levine, S.; and Darrell, T. 2018. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*.

Goecks, V. G.; Gremillion, G. M.; Lawhern, V. J.; Valasek, J.; and Waytowich, N. R. 2020. Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 465–473.

Guez, A.; Mirza, M.; Gregor, K.; Kabra, R.; Racanière, S.; Weber, T.; Raposo, D.; Santoro, A.; Orseau, L.; Eccles, T.; et al. 2019. An investigation of model-free planning. In *International Conference on Machine Learning*, 2464–2473. PMLR.

Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; et al. 2018. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Hu, Y.; Wang, W.; Jia, H.; Wang, Y.; Chen, Y.; Hao, J.; Wu, F.; and Fan, C. 2020. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33: 15931–15941.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1): 97–109.

Ladosz, P.; Weng, L.; Kim, M.; and Oh, H. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85: 1–22.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

Loynd, R.; Fernandez, R.; Celikyilmaz, A.; Swaminathan, A.; and Hausknecht, M. 2020. Working memory graphs. In *International conference on machine learning*, 6404–6414. PMLR.

Moerland, T. M.; Broekens, J.; and Jonker, C. M. 2020. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven Exploration by Self-supervised Prediction. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 2778–2787. PMLR.

Pignat, E.; and Calinon, S. 2019. Bayesian Gaussian mixture model for robotic policy imitation. *IEEE Robotics and Automation Letters*, 4(4): 4452–4458.

Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Jimenez Rezende, D.; Puigdomènech Badia, A.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, 30.

Reddy, S.; Dragan, A. D.; and Levine, S. 2019. SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schrader, M.-P. B. 2018. gym-sokoban. https://github.com/mpSchrader/gym-sokoban.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020a. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020b. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Trott, A.; Zheng, S.; Xiong, C.; and Socher, R. 2019. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in Neural Information Processing Systems*, 32.

Vecerik, M.; Hester, T.; Scholz, J.; Wang, F.; Pietquin, O.; Piot, B.; Heess, N.; Rothörl, T.; Lampe, T.; and Riedmiller, M. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

Wilcox, A.; Balakrishna, A.; Dedieu, J.; Benslimane, W.; Brown, D.; and Goldberg, K. 2022. Monte carlo augmented actor-critic for sparse reward deep reinforcement learning from suboptimal demonstrations. *Advances in Neural Information Processing Systems*, 35: 2254–2267.

Appendix A. Minigrid examples

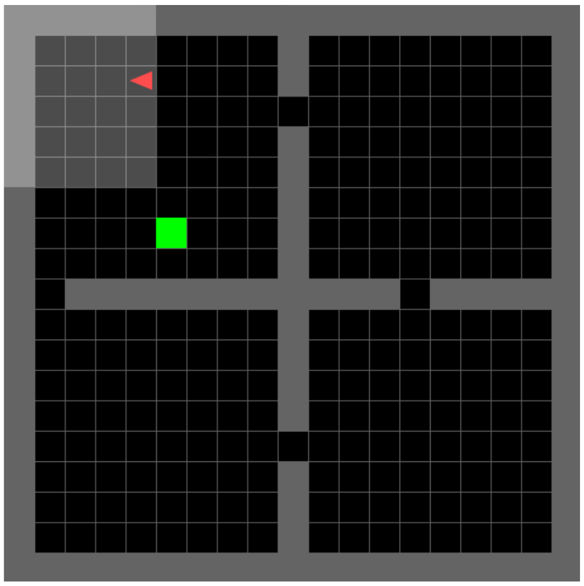
Figure 2: Example of a grid for *MiniGrid-FourRooms-v0*. The agent is indicated by the red pointer.

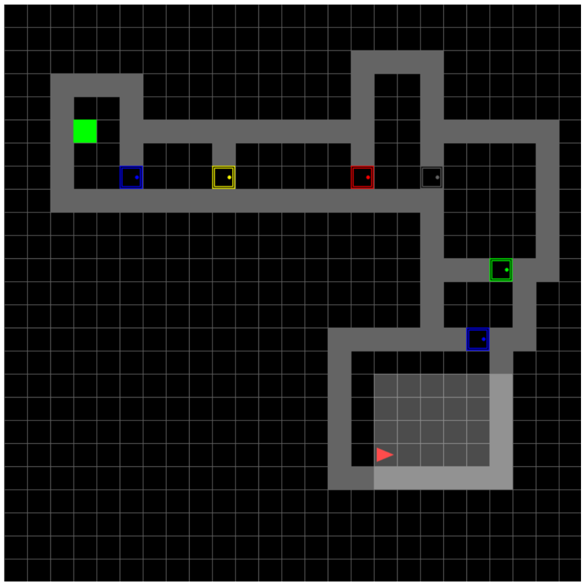
Figure 3: Example of a grid for *MiniGrid-MultiRoom-N6-v0*. The agent is indicated by the red pointer.

Appendix B. Sokoban and deadlocks

A deadlock occurs when a box is not on a goal cell and it can no longer be moved. Because there is no “pull” action, this can happen when it is in a corner. It then becomes impossible for the agent to move to the other side (which is occupied by a wall or another box) to push it in the desired direction. There is also a concept of “soft deadlock” where a block can be pushed alongside a wall, but the goal cell is not along that wall. In that case, it can be impossible for the agent to push the block away from the wall and onto the desired goal cell. See Fig. 4 for examples of deadlocks.

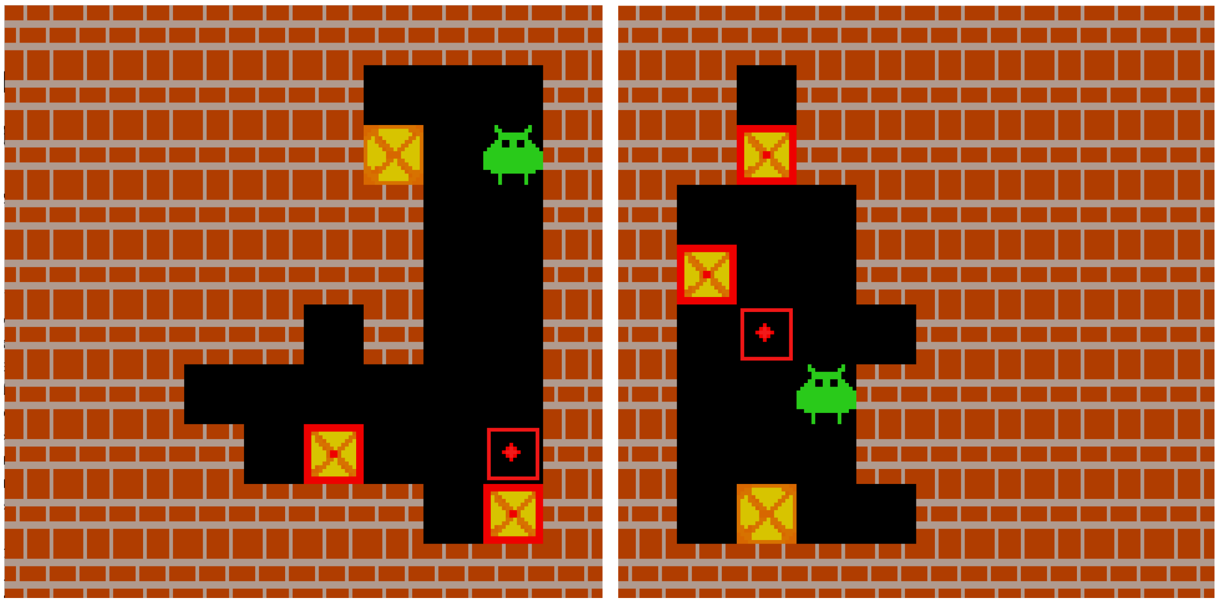

Figure 4: Sokoban grid examples. The green creature is the agent, yellow squares with an 'X' are boxes, red squares with a dot are goals. **Left**: deadlock due to a box stuck in a corner. **Right**: soft deadlock on the bottom wall, with a box that can never reach the remaining unoccupied goal cell.