HOW SHOULD I PLAN? A PERFORMANCE COMPARISON OF DECISION-TIME VS. BACKGROUND PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In model-based reinforcement learning, an agent can leverage a learned model to improve its way of behaving in different ways. Two prevalent approaches are decision-time planning and background planning. In this study, we are interested in *understanding* under what conditions and in which settings one of these two planning styles will perform better than the other in domains that require fast responses. After viewing them through the lens of dynamic programming, we first consider the classical instantiations of these planning styles and provide theoretical results and hypotheses on which one will perform better in the pure planning, planning & learning, and transfer learning settings. We then consider the modern instantiations of these planning styles and provide hypotheses on which one will perform better in the last two of the considered settings. Lastly, we perform several illustrative experiments to empirically validate both our theoretical results and hypotheses. Overall, our findings suggest that even though decision-time planning does not perform as well as background planning in their classical instantiations, in their modern instantiations, it can perform on par or better than background planning in both the planning & learning and transfer learning settings.

1 INTRODUCTION

It has long been argued that, in order for reinforcement learning (RL) agents to adapt to a variety of changing tasks, they should be able to learn a model of their environment, which allows for counterfactual reasoning and fast re-planning [11]. Although this is a widely-accepted view in the RL community, the question of *how* to leverage a learned model to perform planning in the first place does not have a widely-accepted and clear answer. In model-based RL, the two prevalent planning styles are decision-time (DT) and background (B) planning [17], where the agent mainly plans in the moment and in parallel to its interaction with the environment, respectively. Even though these two planning styles have been developed with different assumptions and application domains in mind, i.e., DT planning algorithms [20; 21; 13; 14] under the assumption that the exact model of the environment is known and for domains that allow for certain computational budgets, such as board games, and B planning algorithms [15; 16; 26; 6] under the assumption that the exact model is unknown and for domains that usually require fast responses, such as basic gridworlds and video games, recently, with the introduction of the ability to learn a model through pure interaction [12], DT planning algorithms have been applied to the same domains as their B planning counterparts (see e.g., [12; 7]). However, it still remains unclear under *what* conditions and in *which* settings one of these planning styles will perform better than the other in these fast-response-requiring domains.

To clarify this, we first start by abstracting away from the specific implementation details of the two planning styles and view them in a unified way through the lens of dynamic programming. Then, we consider the classical instantiations (CI) of these planning styles and based on their dynamic programming interpretations, provide theoretical results and hypothesis on which one will perform better in the pure planning (PP), planning & learning (P&L), and transfer learning (TL) settings. We then consider the modern instantiations (MI) of these two planning styles and based on both their dynamic programming interpretations and implementation details, provide hypotheses on which one will perform better in the P&L and TL settings. Lastly, we perform illustrative experiments with both instantiations of these planning styles to empirically validate our theoretical results and hypotheses. Overall, our results suggest that even though DT planning does not perform as well as B planning in their CIs, due to (i) the improvements in the way planning is performed, (ii) the usage

of only real experience in the updates of the value estimates, and (iii) the ability to improve upon the previously obtained policy at test time, the MIs of it can perform on par or better than their B planning counterparts in both the P&L and TL settings. We hope that our findings will help the RL community in developing a better understanding of the two planning styles and stimulate research in improving of them in potentially interesting ways.

Related Work. Our analysis is mostly related to the recent monograph of Bertsekas [2] in which the recent successes of AlphaZero [14], a DT planning algorithm, are viewed through the lens of dynamic programming. However, we take a broader perspective that encompasses both DT and B planning and also consider the case in which the model is learned rather than given. Another closely related study is Hamrick et al. [7] which informally relates MuZero [12], another DT planning algorithm, to various other DT and B planning algorithms in the literature. Our study can be viewed as a study that formalizes the relation between the two planning styles. There have also been studies that empirically compare the performances of various DT and B planning algorithms on continuous control domains in the P&L setting [24], and on minigrid environments in specific TL settings [25]. However, none provide a general understanding of when will one planning style perform better than the other.

2 BACKGROUND

In RL [17], an agent A interacts with its environment E through a sequence of actions to maximize its long-term cumulative reward. Here, the environment is usually modeled as a Markov decision process $E = (\mathcal{S}_E, \mathcal{A}_E, p_E, r_E, d_E, \gamma)$, where $\mathcal{S}_E$ and $\mathcal{A}_E$ are the (finite) set of states and actions, $p_E : \mathcal{S}_E \times \mathcal{A}_E \times \mathcal{S}_E \to [0, 1]$ is the transition distribution, $r_E : \mathcal{S}_E \times \mathcal{A}_E \times \mathcal{S}_E \to \mathbb{R}$ is the reward function, $d_E : \mathcal{S}_E \to [0, 1]$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. At each time step t, after taking an action $a_t \in \mathcal{A}_E$, the environment's state transitions from $s_t \in \mathcal{S}_E$ to $s_{t+1} \in \mathcal{S}_E$, and the agent receives an observation $o_{t+1} \in \mathcal{O}_E$ and an immediate reward r_t. We assume that the observations are generated by a deterministic procedure $\psi : \mathcal{S}_E \to \mathcal{O}_E$, unknown to the agent. As the agent usually does not have access to the states in $\mathcal{S}_E$ a priori, and as the observations in $\mathcal{O}_E$ are usually very high-dimensional, it has to operate on its own state space $\mathcal{S}_A$, which is generated by its own adaptable (or sometimes a priori fixed) value encoder $\phi : \mathcal{O}_E \to \mathcal{S}_A$. The goal of the agent is to jointly learn a value encoder ϕ and a value estimator $Q : \mathcal{S}_A \times \mathcal{A}_E \to \mathbb{R}$ that induces a policy $\pi : \mathcal{S}_A \times \mathcal{A}_E \to [0, 1] \in \mathbb{\Pi}$, where $\mathbb{\Pi} \equiv \{\pi | \pi : \mathcal{S}_A \times \mathcal{A}_E \to [0, 1]\}$, maximizing $E_{\pi, p_E}[\sum_{t=0}^{\infty} \gamma^t r_E(S_t, A_t, S_{t+1}) | S_0 \sim d_E]$. For convenience, we will refer to the composition of ϕ and Q as simply the value estimator.

Model-Based RL. One way of achieving this goal is through the use model-based RL (MBRL) methods. In MBRL, there are two (alternating) phases: the learning and planning phases. In the learning phase, the gathered experience is mainly used in jointly learning an adaptable (or sometimes a priori fixed) model encoder $\varphi : \mathcal{O}_E \to \mathcal{S}_M$ and a model $m \equiv (p_M, r_M, d_M) \in \mathcal{M}$, where $\mathcal{M} \equiv \{(p_M, r_M, d_M) | p_M : \mathcal{S}_M \times \mathcal{A}_E \times \mathcal{S}_M \to [0, 1], r_M : \mathcal{S}_M \times \mathcal{A}_E \times \mathcal{S}_M \to \mathbb{R}, d_M : \mathcal{S}_M \to [0, 1]\}$ and $\mathcal{S}_M$ is the state space of the agent's model, and optionally, the experience may also be used in improving the value estimator. Again, for convenience, we will refer to the composition of φ and m as simply the model. In the planning phase, the learned model m is then used for simulating experience, either to be used alongside real experience, or just to be used in selecting actions at decision time. Note that in general $\varphi \neq \phi$ and thus $\mathcal{S}_M \neq \mathcal{S}_A$. However, in these cases, we assume that the agent has access to a deterministic function $\rho : \mathcal{S}_M \to \mathcal{S}_A$ that allows for going from $\mathcal{S}_M$ to $\mathcal{S}_A$. We also assume that $\mathcal{S}_E \subseteq \mathcal{S}_M$, which implies that the agent's model is, in principle, capable of exactly modeling the environment, though this may be very hard in practice.

Planning Styles in MBRL. In MBRL, planning is performed in two different styles: (i) DT planning, and (ii) B planning (see Ch. 8 of [17]). DT planning is performed as a computation whose output is the selection of a single action for the current state. This is often done by unrolling the model forward from the current state to compute local value estimates, which are then usually discarded after action selection. Here, planning is performed independently for *every* encountered state and it is mainly performed in an *online* fashion, though it may also contain offline components. In contrast, B planning is performed by continually improving a cached value estimator, on the basis of simulated experience from the model, often in a global manner. Action selection is then quickly done by querying the estimator at the current state. Unlike DT planning, B planning is often performed in a purely *offline* fashion, in parallel to the agent-environment interaction, and thus is *not* necessarily focused on the

current state: well before action selection for any state, planning plays its part in improving the value estimates in many other states. For convenience, in this study, we will refer to all MBRL algorithms that have an online planning component as DT planning algorithms (see e.g., [20; 21; 13; 14; 12; 25]), and will refer to the rest as B planning algorithms (see e.g., [15; 16; 26; 6; 25]). Note that, regardless of the style, any type of planning can be viewed as a procedure $f : (\mathcal{M}, \Pi) \rightarrow \Pi$ that takes a model m and a policy π^i as input and returns an improved policy π^o_m, according to m, as output.

Algorithms within the Two Planning Styles. Starting with DT planning, depending on how much search is performed, DT planning algorithms can be studied under three main categories: DT planning algorithms (i) that perform no search (see e.g., [21] and Alg. 1), (ii) that perform pure search (see e.g., [4] and Alg. 2), and (iii) that perform some amount of search (see e.g., [13; 14; 12] and Alg. 5). In the first two, planning is performed by just running pure rollouts with a fixed or improving policy (see Fig. 5a), and by purely performing search (see Fig. 5b), respectively. In the last one, planning is performed by first performing some amount of search and then either by running rollouts with a fixed or improving policy, by bootstrapping on the cached value estimates of a fixed or improving policy, or by doing both (see Fig. 5c & 5d). Note that while the CIs of DT planning fall within the first two categories, the MIs of it usually fall within the last one. Also note that, while planning is performed with only a single parametric model in the first two categories, it is usually performed with both a parametric and non-parametric (usually a replay buffer, see [22]) model in the last one (see e.g., [12] and Alg. 5). See [2] for more details on the different categories of DT planning. Moving on to B planning, as all B planning algorithms (see e.g., Alg. 3, 4, 6) perform planning by periodically improving a cached value estimator throughout the model learning process, we do not study them under different categories. However, we again note that, while some B planning algorithms perform planning with a single parametric model (see e.g., Alg. 3 & 4), some perform planning with both a parametric and non-parametric (again usually a replay buffer) model (see e.g., Alg. 6).

3 A Unified View of the Two Planning Styles

In this section, we abstract away from the specific implementation details, such as whether policy improvement is done locally or globally, or whether planning is performed in an online or offline manner, and view the two planning styles in a unified way through the lens of dynamic programming [3]. More specifically, we view DT and B planning through the lens of policy iteration (PI). [1] In this framework, DT planning algorithms that perform no search can be considered as performing one-step PI at every encountered state [2], on top of a fixed or improving policy, as they compute π^o_m by first running many rollouts with a fixed or improving π^i in m to evaluate the current state (policy evaluation), and then by selecting the most promising action (policy improvement). Similarly, DT planning algorithms that perform pure search can be considered as performing PI until convergence (which we call full PI) at every encountered state as they disregard π^i and compute π^o_m by first performing exhaustive search in m to obtain the optimal values at the current state, and then by selecting the most promising action. Finally, DT planning algorithms that perform some amount of search can be considered as performing something between one-step and full PI at every encountered state, on top of a fixed or improving policy, as they are just a mixture of DT planning algorithms that perform no search and pure search. Hence, depending on how much search is performed, DT planning algorithms in general can be viewed as going between the spectrum of performing one-step PI and full PI at every encountered state. Similarly, all B planning algorithms can also be considered as performing either a fraction of one-step PI, full PI, or something in between, as they compute π^o_m by periodically improving a fixed or improving π^i on the basis of simulated experience from m, either for a single time step, until convergence, or somewhere in between. Thus, depending on much planning is performed, B planning algorithms in general can also be viewed as interpolating between performing one-step PI and full PI.

Finally, note that, in Sec. 2, we have pointed out that some DT and B planning algorithms perform planning with both a parametric and non-parametric model, which can make it hard for them to be viewed through our proposed framework. However, if one considers the two separate models as a single combined model, then these algorithms can also be viewed straightforwardly in our proposed framework (see App. F).

[1] Note that we choose PI over value iteration as it better describes the underlying working principles of DT planning (see [2]), and it is also useful in understanding the underlying working principles of B planning.

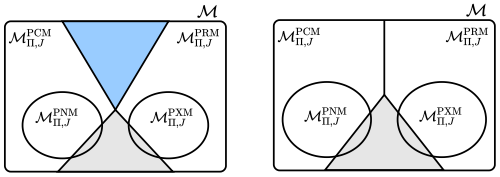
4 DECISION-TIME VS. BACKGROUND PLANNING

In this study, we are interested in understanding under what conditions and in which settings one planning style will perform better than the other. Thus, we start by defining a performance measure that will be used in comparing the two planning styles of interest. Given an arbitrary model $m = (p, r, d) \in \mathcal{M}$, let us define the performance of an arbitrary policy $\pi \in \mathbb{\Pi}$ in it as follows:

$$J^{\pi}_{m} \equiv E_{\pi,p}[\textstyle\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t, S_{t+1}) | S_0 \sim d]. \tag{1}$$

Note that J^{π}_{m} corresponds to the expected *discounted* return of a policy π in model m. Next, we consider the conditions under which the comparisons will be made: we are interested in both simple scenarios in which the value estimators and models are both represented as a table, and in complex ones in which at least the value estimators or models are represented using function approximation.

Partitioning of the Model Space. We now present a way to partition the space of agent models $\mathcal{M}$ such that one planning style is guaranteed to perform on par or better than the other. Let us start by defining m^* to be the exact model of the environment. Note that $m^* \in \mathcal{M}$, as we assumed that $\mathcal{S}_E \subseteq \mathcal{S}_M$ (see Sec. 2). Then, given a policy set $\Pi \subseteq \mathbb{\Pi}$, containing at least two policies, and a performance measure J, defined as in (1), depending on the relative performances of the policies in it and in m^*, a model $m \in \mathcal{M}$ can belong to one of the following main classes:



(a) General (b) Interest

Figure 1: (a) The general partitioning and (b) the partitioning of interest of $\mathcal{M}$, for a given Π and J. The gray and blue regions indicate $\mathcal{M}^{\text{PCM}}_{\Pi,J} \cap \mathcal{M}^{\text{PRM}}_{\Pi,J}$ and $\mathcal{M} \setminus (\mathcal{M}^{\text{PCM}}_{\Pi,J} \cup \mathcal{M}^{\text{PRM}}_{\Pi,J})$, respectively.

Definition 1 (PCM)**.** *Given $\Pi \subseteq \mathbb{\Pi}$ and J, let $\mathcal{M}^{\text{PCM}}_{\Pi,J} \equiv \{m \in \mathcal{M} \mid J^{\pi^i}_{m^*} \lesseqgtr J^{\pi^j}_{m^*} \; \forall \pi^i, \pi^j \in \Pi$ satisfying $J^{\pi^i}_{m} \gtreqless J^{\pi^j}_{m}\}$. We say that each $m \in \mathcal{M}^{\text{PCM}}_{\Pi,J}$ is a* performance-contrasting model (PCM) *of m^* w.r.t. Π and J.*

Definition 2 (PRM)**.** *Given $\Pi \subseteq \mathbb{\Pi}$ and J, let $\mathcal{M}^{\text{PRM}}_{\Pi,J} \equiv \{m \in \mathcal{M} \mid J^{\pi^i}_{m^*} \gtreqless J^{\pi^j}_{m^*} \; \forall \pi^i, \pi^j \in \Pi$ satisfying $J^{\pi^i}_{m} \gtreqless J^{\pi^j}_{m}\}$. We say that each $m \in \mathcal{M}^{\text{PRM}}_{\Pi,J}$ is a* performance-resembling model (PRM) *of m^* w.r.t. Π and J.*

Informally, given any two policies in Π, and J, a model m is a PCM of m^* iff the policy that performs on par or better in it performs on par or worse in m^*, and it is a PRM of m^* iff the policy that performs on par or better in it also performs on par or better in m^*. Note that m can both be a PCM and a PRM of m^* iff the two policies perform on par in both m and m^*. If Π contains at least one of the optimal policies for m, then m can also belong to one of the following specialized classes:

Definition 3 (PNM)**.** *Given $\Pi \subseteq \mathbb{\Pi}$ and J, let $\mathcal{M}^{\text{PNM}}_{\Pi,J} \equiv \{m \in \mathcal{M}^{\text{PCM}}_{\Pi,J} \mid J^{\pi^*_m}_{m^*} = \min_{\pi \in \mathbb{\Pi}} J^{\pi}_{m^*} \; \forall \pi^*_m \in \Pi\}$, where π^*_m denotes the optimal policies in m. We say that each $m \in \mathcal{M}^{\text{PNM}}_{\Pi,J}$ is a* performance-minimizing model (PNM) *of m^* w.r.t. Π and J.*

Definition 4 (PXM)**.** *Given $\Pi \subseteq \mathbb{\Pi}$ and J, let $\mathcal{M}^{\text{PXM}}_{\Pi,J} = \{m \in \mathcal{M}^{\text{PRM}}_{\Pi,J} \mid J^{\pi^*_m}_{m^*} = \max_{\pi \in \mathbb{\Pi}} J^{\pi}_{m^*} \; \forall \pi^*_m \in \Pi\}$, where π^*_m denotes the optimal policies in m. We say that each $m \in \mathcal{M}^{\text{PXM}}_{\Pi,J}$ is a* performance-maximizing model (PXM) *of m^* w.r.t. Π and J.*

Informally, given a subset of Π that contains the optimal policies for a model m, and J, m is a PNM of m^* iff all of the optimal policies result in the worst possible performance in m^*, and it is a PXM of m^* iff all them result in the best possible performance in m^*. Note that all definitions above are agnostic to how the models are represented.

Fig. 1a illustrates how $\mathcal{M}$ is partitioned for a given Π and J. Note that for a fixed J, the relative sizes of the model classes solely depend on Π. For instance, as Π gets larger, the relative sizes of $\mathcal{M}^{\text{PCM}}_{\Pi,J}$ and $\mathcal{M}^{\text{PRM}}_{\Pi,J}$ shrink, because with every policy that is added to Π, the number of criteria that a model must satisfy to be a PCM or PRM increases, which reduces the odds of an arbitrary model in $\mathcal{M}$ being in $\mathcal{M}^{\text{PCM}}_{\Pi,J}$ or $\mathcal{M}^{\text{PRM}}_{\Pi,J}$. And, as Π gets smaller, the relative sizes of $\mathcal{M}^{\text{PCM}}_{\Pi,J}$ and $\mathcal{M}^{\text{PRM}}_{\Pi,J}$ grow, and eventually fill up the entire space, when Π contains only two policies. Fig. 1b illustrates the partitioning in this scenario. Since we are only interested in comparing the policies of two planning styles, the Π of interest has a size of two, and thus we have a partitioning as in Fig. 1b.

4.1 CLASSICAL INSTANTIATIONS OF THE TWO PLANNING STYLES

For easy analysis, we start by considering the CIs of the two planning styles in which both the value estimators and models are represented as a table. More specifically, for DT planning we study a version of the OMCP algorithm of Tesauro & Galperin [21] in which a parametric model is learned from experience (see Alg. 1), and for B planning we study a simplified version of the Dyna-Q algorithm of Sutton [15; 16] in which planning is performed until convergence with every model in the model learning process blue (see Alg. 4). See App. B for a discussion on why we consider these versions. Note that these two algorithms can be considered as performing one-step PI and full PI on top of a fixed policy, respectively (see Sec. 3). Also note that, although we only consider these specific instantiations, as long as the value estimators of both planning styles are represented as a table and DT planning corresponds to taking a smaller or on par policy improvement step than B planning, which is the case in most CIs, the results that we derive in this section would hold regardless of the choice of instantiation. Before considering different settings, let us define the following policies that will be useful in referring to the input and output policies of the two planning styles:

Definition 5 (Base, Rollout [2] & CE [9] Policies)**.** *The* base policy $\pi^b \in \mathbb{\Pi}$ *is the policy used in initiating PI. Given a fixed or improving base policy* π^b *and a model* $m \in \mathcal{M}$*, the* rollout policy $\pi^r_m \in \mathbb{\Pi}$ *is the policy obtained after one-step of PI on top of* π^b *in* m*, and the* certainty-equivalence (CE) policy $\pi^{ce}_m \in \mathbb{\Pi}$ *is the policy obtained after full PI on top of* π^b *in* m*.*

PP Setting. We start by considering the PP setting in which the agent is directly provided with a model. In this setting, we can prove the following statements:

Proposition 1. *Let* $m \in \mathcal{M}$ *be a PCM of* m^* *w.r.t.* $\Pi = \{\pi^r_m, \pi^{ce}_m\} \subseteq \mathbb{\Pi}$ *and* J*. Then,* $J^{\pi^r_m}_{m^*} \geq J^{\pi^{ce}_m}_{m^*}$.

Proposition 2. *Let* $m \in \mathcal{M}$ *be a PRM of* m^* *w.r.t.* $\Pi = \{\pi^r_m, \pi^{ce}_m\} \subseteq \mathbb{\Pi}$ *and* J*. Then,* $J^{\pi^{ce}_m}_{m^*} \geq J^{\pi^r_m}_{m^*}$.

Due to space constraints, we defer all the proofs to App. C. Prop. 1 & 2 imply that, given $\Pi = \{\pi^r_m, \pi^{ce}_m\}$ and J, although DT planning will perform on par or better than B planning when the provided model m is a PCM, it will perform on par or worse when m is a PRM. Note that these results would not be guaranteed to hold if function approximation was introduced in the value estimator representations, as in this case, there would be no guarantee that full PI will result in a better policy than one-step PI in m [3]. However, if one were to use approximators with good generalization capabilities (GGC), i.e., approximators that assign the same value to similar observations, we would expect a similar performance trend to hold.

P&L Setting. In the P&L setting, the model has to be learned from the experience gathered by the agent. In this scenario, as different policies are likely to be used in the model learning process, the encountered models of the two planning styles, which we denote as $\bar{m} \in \mathcal{M}$ and $\bar{\bar{m}} \in \mathcal{M}$ for DT and B planning, respectively, are also likely to be different. Thus, as they require the two planning styles to have access to the same model, the results of Prop. 1 & 2 are not valid in the P&L setting. However, if the model of B planning is a PNM or a PXM, we can prove the following statements:

Proposition 3. *Let* $\bar{m} \in \mathcal{M}$ *be a PCM or a PRM of* m^* *w.r.t.* $\bar{\Pi} = \{\pi^r_{\bar{m}}, \pi^{ce}_{\bar{m}}\} \subseteq \mathbb{\Pi}$ *and* J*, and let* $\bar{\bar{m}} \in \mathcal{M}$ *be a PNM of* m^* *w.r.t.* $\bar{\bar{\Pi}} = \{\pi^r_{\bar{\bar{m}}}, \pi^{ce}_{\bar{\bar{m}}}\} \subseteq \mathbb{\Pi}$ *and* J*. Then,* $J^{\pi^r_{\bar{\bar{m}}}}_{m^*} \geq J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*}$.

Proposition 4. *Let* $\bar{m} \in \mathcal{M}$ *be a PCM or a PRM of* m^* *w.r.t.* $\bar{\Pi} = \{\pi^r_{\bar{m}}, \pi^{ce}_{\bar{m}}\} \subseteq \mathbb{\Pi}$ *and* J*, and let* $\bar{\bar{m}} \in \mathcal{M}$ *be a PXM of* m^* *w.r.t.* $\bar{\bar{\Pi}} = \{\pi^r_{\bar{\bar{m}}}, \pi^{ce}_{\bar{\bar{m}}}\} \subseteq \mathbb{\Pi}$ *and* J*. Then,* $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} \geq J^{\pi^r_{\bar{\bar{m}}}}_{m^*}$.

Prop. 3 & 4 imply that, given $\bar{\Pi} = \{\pi^r_{\bar{m}}, \pi^{ce}_{\bar{m}}\}$, $\bar{\bar{\Pi}} = \{\pi^r_{\bar{\bar{m}}}, \pi^{ce}_{\bar{\bar{m}}}\}$ and J, although DT planning will perform on par or better than B planning when $\bar{\bar{m}}$ is a PNM, it will perform on par or worse when $\bar{\bar{m}}$ is a PXM. While the former result can be relevant in the initial phases of B planning's model learning process, the latter one is most likely to be relevant in the final phases of this process, when $\bar{\bar{m}}$ becomes a PXM. Note that since the models are represented as tables, the learned models of both planning styles are guaranteed to become PXMs in the limit, as we know that in the worst case, with sufficient exploration, the models will, in the limit, converge to m^* [17]. However, note that this would not be guaranteed if function approximation was used in the model representations. Lastly, even if $\bar{\bar{m}}$ starts as a PNM and eventually becomes a PXM, the results of Prop. 3 & 4 would not be guaranteed to hold if function approximation was used in the value estimator representations, due to the reason discussed in the PP setting. However, if one were to use approximators with GGC, we would again expect a similar performance trend to hold.

TL Setting. Although there are many different settings in TL [19], for easy analysis, we start by considering the simplest one in which there is only one training task and one subsequent test task,

differing only in their reward functions, and in which the agent's transfer ability is measured by how fast it adapts to the test tasks after being trained on the training tasks. In this setting, we would expect the results of the P&L setting to hold directly, as instead of a single one, there are now two consecutive P&L settings.

4.2 Modern Instantiations of the Two Planning Styles

We now consider the MIs of the two planning styles in which both the value estimators and models are represented with neural networks (NN). More specifically, we study the DT and B planning algorithms in Zhao et al. [25] (see Alg. 5 & 6) as they are reflective of many of the properties of their state-of-the-art counterparts (see e.g., [12; 26; 6]) and their code is publicly available. See App. E for a broader discussion on why we choose these algorithms. Here, as the DT planning algorithm performs planning by first performing some amount of search with a parametric model and then by bootstrapping on the value estimates of a continually improving policy, it can be considered as performing more than one-step PI on top of an improving policy in a combined model $\bar{m}_c$ (see Sec. 3). And, as the B planning algorithm performs planning by continually improving a value estimator at every time step with both a parametric and non-parametric (a replay buffer) model, it can be viewed as performing something between a fraction of one-step PI and full PI on top of an improving policy with a combined model $\bar{\bar{m}}_c$ (see Sec. 3). However, if $\bar{\bar{m}}_c$ converges, it can be viewed as performing full PI, as in this case the continual improvements to the value estimator with the converged $\bar{\bar{m}}_c$ would eventually lead to an improvement that is equivalent to performing full PI. Note that although we only consider these specific instantiations, the hypotheses we provide in this section are generally applicable to most state-of-the-art MBRL algorithms, as the algorithms in [25] are reflective of many of their properties (see App. E).

P&L Setting. To ease the analysis, let us start by considering a simplified scenario in which both the value estimators and models of the MIs of the two planning styles are represented as a table. Let us also define the *improved rollout policy* to be the policy $\pi^{r+}_m \in \mathbb{\Pi}$ obtained after performing more than one-step PI, with the exact number not being important, on top of a base policy $\pi^b \in \mathbb{\Pi}$ in model m, and let us also refer to the policies generated by the MIs of DT and B planning with models $\bar{m}_c$ and $\bar{\bar{m}}_c$ on top of an improving base policy π^b as $\pi^{r+}_{\bar{m}_c}$ and $\pi^{ce}_{\bar{\bar{m}}_c}$, respectively. Then, using $\pi^{r+}_{\bar{m}_c}$ and $\pi^{ce}_{\bar{\bar{m}}_c}$ in place of $\pi^{r}_{\bar{m}}$ and $\pi^{ce}_{\bar{\bar{m}}}$, respectively, and under the assumption that $\bar{\bar{m}}_c$ converges, we would expect the formal statements of the P&L setting of Sec. 4.1 to hold exactly as DT planning still corresponds taking a smaller or on par policy improvement step than B planning. However, as DT planning now corresponds to performing more than one-step PI, we would expect the performance gap between the two planning styles to reduce in their MIs. Moreover, we would expect this gap to gradually close if both $\bar{m}_c$ and $\bar{\bar{m}}_c$ become, and remain as, PXMs, as the use of an improving policy for DT planning would result in a continually improving performance that gets closer to the one of B planning.

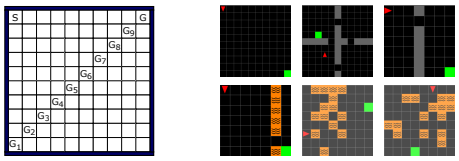
Coming back to our original scenario in which both the value estimators and models of the MIs of the two planning styles are represented with NNs, we would expect a similar performance trend to hold as NNs are approximators that have GGC. However, this expectation is solely based on the dynamic programming interpretations of the two planning styles and thus does not take into account the implementation details of them, which can also play an important role on how the two planning styles will perform against each other in their MIs. In their MIs, both planning styles perform planning by unrolling NN-based parametric models which are known to easily lead to compounding model errors (CME, [18]). Thus, obtaining combined models that are PXMs becomes quite difficult, if not impossible, for both planning styles. Even though this problem can significantly be mitigated for both of them by unrolling the models for only a few time steps and then bootstrapping on the value estimators for the rest, B planning can also suffer from updating its value estimator with the potentially harmful simulated experience generated by its NN-based parametric model (see e.g., [22; 8]), which can slowdown, or even prevent, it in reaching optimal (or close to optimal) performance. Note that this is not a problem in DT planning as its value estimator is only updated with real experience. Thus, based on these observations, we hypothesize that compared to DT planning, it is likely for B planning to suffer more in reaching optimal (or close to optimal) performance in their MIs.

TL Setting. We now consider two common TL settings that are both more challenging than the TL setting in Sec. 4.1. In these settings, there is a distribution of training tasks and test tasks, differing only in their observations. In the first one, the agent's transfer ability is measured by how fast it adapts to the test tasks after being trained on the training tasks (see e.g., [23]), and in the second

one, this ability is measured by its instantaneous performance on the test tasks as it gets trained on the training tasks, also known as "zero-shot transfer" in the literature (see e.g., [25; 1]). In both TL settings, we would again expect B planning to suffer more in reaching the optimal (or close to optimal) performance on the training tasks because of the reasons discussed in the P&L setting. Additionally, in the first setting, after the tasks switch from the training tasks to the test tasks, we would expect B planning to suffer more in the adaptation process, as its parametric model, learned on the training tasks, would keep generating experience that resembles the training tasks until it adapts to the test tasks, which in the meantime can lead to harmful updates to its value estimator. Also, in the second setting, if the learned parametric model of DT planning is capable of simulating at least a few time steps of the test tasks, and if the learned policies of the both planning styles perform similarly on the test tasks, we would expect DT planning to perform better on the test tasks, as at test time, it would be able to improve upon the policy obtained during training by performing online planning. Note that, this is not possible for B planning, as it performs planning in an offline fashion and thus requires additional interaction with the test tasks to improve upon the policy obtained during training.

5 EXPERIMENTS

We now perform illustrative experiments to validate the formal statements and hypotheses presented in Sec. 4. The experimental details can be found in App. G & H, respectively. We perform experiments on both the Simple Gridworld (SG) environment and on environments from MiniGrid (MG, [5]) (see Fig. 2), as the optimal policies in these environments are easy to learn and they allow for designing controlled experiments that are helpful in answering the questions of interest to this study. In the SG environment, the agent spawns in state S and has to navigate to the goal state depicted by G. In the MG environments, the agent, depicted in red, has to navigate to the green goal cell, while avoiding the orange lava cells (if there are any). More details can be found in App. G. Note that while $\mathcal{O}_E = \mathcal{S}_E$ in the SG environment, $\mathcal{O}_E \neq \mathcal{S}_E$ in the MG environments.



(a) SG Environment (b) MiniGrid Environments

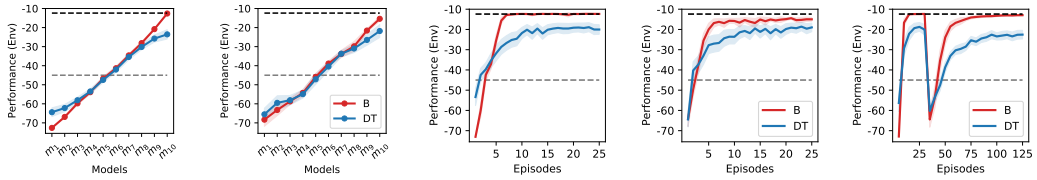
Figure 2: (a) The SG environment and (b) MG environments: (top row) Empty 10x10, FourRooms, SCS9N1, (bottom row) LCS9N1, RDS Train, RDS Test.

5.1 EXPERIMENTS WITH CLASSICAL INSTANTIATIONS

In this section, we perform experiments with the CIs of DT and B planning (see Alg. 1 & 4) on the SG environment to empirically validate our theoretical results and hypotheses in Sec. 4.1. In addition to the scenario in which both the value estimators and models are represented as tables (where $\phi = \varphi$ are both identity functions, implying $\mathcal{S}_A = \mathcal{S}_M = \mathcal{O}_E$), we also consider the one in which only the model is represented as a table (where only φ is an indentity , implying $\mathcal{S}_M = \mathcal{O}_E$). In the latter scenario, we use state aggregation (SA) in the value estimator representation, i.e., ϕ is a state aggregator. More on the implementation details of these CIs can be found in App. G.1.

PP Experiments. According to Prop. 1 & 2, although DT planning is guaranteed to perform on par or better than B planning when the provided model is a PCM, it is guaranteed to perform on par or worse when the provided model is a PRM. To empirically verify this, we provided the two planning styles with a series of hand-designed tabular models that first start with the PNM m_1 with goal state G_1 (see Fig. 2a), and then gradually move towards the PXM m_{10} with goal state G, by first becoming PCMs $\{m_i\}_{i=2}^{5}$, and then by becoming PRMs $\{m_j\}_{j=6}^{9}$, with goal states $\{\mathrm{G}_n\}_{n=2}^{9}$, respectively (see App. G.1 for more details on these models). After planning was performed with each of these models, we evaluated the resulting output policies in the environment. Results are shown in Fig. 3a. We can indeed see that although DT planning performs better when the provided model is a PCM (or a PNM), it performs worse when the provided model is a PRM (or a PXM). To see if similar results would hold with approximators that have GGC, we also performed the same experiment with SA used in the value estimator representation. Results in Fig. 3b show that a similar trend holds in this case as well.

P&L Experiments. Prop. 3 & 4 imply that, although DT planning is guaranteed to perform on par or better than B planning when the model of B planning is a PNM, it is guaranteed to perform on par or worse when the model of B planning is a PXM. To empirically verify this, we initialized the tabular models of both planning styles as hand-designed PNMs (see App. G.1 for the details) and let them



(a) Tabular (b) State Aggregation (c) Tabular (d) State Aggregation (e) Tabular

Figure 3: The performance of the CIs of DT and B planning on the SG environment, in the (a, b) PP, (c, d) P&L, and (e) TL settings with tabular and SA representations. Black & gray dashed lines indicate the performance of the optimal & random policies, respectively. Shaded regions are one standard error over 250 runs.

be updated through interaction to become PXMs. After every episode, we evaluated the resulting output policies in the environment. Results in Fig. 3c show that, as expected, although DT planning performs better when B planning's model is a PNM, it performs worse when B planning's model becomes a PXM. Again, to see whether if similar results would hold with approximators that have GGC, we also performed experiments with SA used in the value estimator representation. Results in Fig. 3d show that a similar trend holds in this case as well.

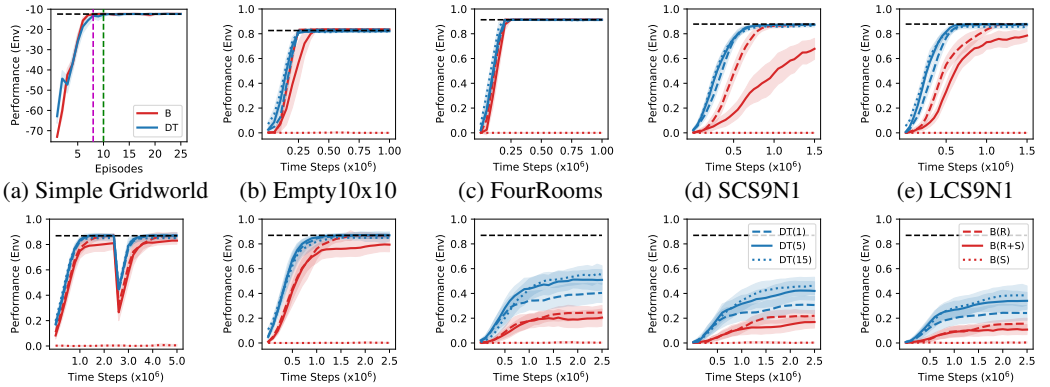
TL Experiments. In Sec. 4.1, we argued that the results of the P&L setting would hold directly in the considered TL setting. For empirical verification, we performed a similar experiment to the one in the P&L setting, in which we initialized the tabular models of both planning styles as hand-designed PNMs and let them be updated to become PXMs. However, differently, after 25 episodes, we now added a subsequent test task with goal state G_1 to the training task (see App. G.1 for the details). In Fig. 3e, we can see that, similar to the P&L setting, before the task changes, DT planning first performs better and but then worse than B planning, and the same happens after the task change. Results in Fig. 13a show that a similar trend also holds when SA used in the value estimator representation.

5.2 Experiments with Modern Instantiations

We now perform experiments with the MIs of DT and B planning to empirically validate our hypotheses in Sec. 4.2. For the experiments with the SG environment, we consider the former scenario in Sec. 5.1, and for the experiments with MG environments, we consider the scenario in which both the value estimators and models are represented with NNs, and in which they share the same encoder (where $\phi = \varphi$ are both learnable parametric functions, implying $\mathcal{S}_A = \mathcal{S}_M \neq \mathcal{O}_E$). More on the implementation details of these MIs can be found in App. G.2.

P&L Experiments. In Sec. 4.2, we argued that if one were to use tabular value estimators and models in the MIs, the results of the P&L section of Sec. 4.1 would hold exactly. Additionally, we also argued that the performance gap between the two planning styles would reduce in their MIs and even gradually close if the combined models of both planning styles become, and remain as, PXMs. To test these, we implemented simplified tabular versions of the MIs of the two planning styles (see Alg. 7 & 8) and compared them on the SG environment. Results are shown in Fig. 4a. As expected, the results of the P&L section of Sec. 4.1 hold exactly, and the performance gap between the two planning styles indeed reduces and gradually closes after the models become, and remain as, PXMs.

We then argued that although the usage of NNs in the representation of the value estimators and models of the two planning styles would not break the performance trend, their implementation details can also play an important role on how they would compare against each other. We hypothesized that although both planning styles would suffer from CMEs, B planning would additionally suffer from updating its value estimator using potentially harmful simulated experience, and thus it is likely to suffer more in reaching optimal (or close to optimal) performance. To test these hypotheses, we compared the MIs of the two planning styles (see Alg. 5 & 6) on several MG environments. In order to test the effect of CMEs in DT planning, we performed experiments in which the parametric model is unrolled for 1, 5, and 15 time steps during search, denoted as DT(1), DT(5) and DT(15). Note that CMEs are not a problem for B planning, as its parametric model is only unrolled for a single time step. Also, in order to test the effect of updating the value estimator with simulated experience in B planning, we performed experiments in which its value estimator is updated with only real, both real and simulated, and only simulated experience, denoted as B(R), B(R+S) and B(S). See App. G.2 for the details of these experiments. The results, in Fig. 4b-4e, show that even though the mean



(a) Simple Gridworld (b) Empty10x10 (c) FourRooms (d) SCS9N1 (e) LCS9N1

(f) RDS Sequential (g) RDS Train (0.35) (h) RDS Test (0.25) (i) RDS Test (0.35) (j) RDS Test (0.45)

Figure 4: The performance of the MIs of DT and B planning in the (a-e) P&L and (f-j) TL settings with (a) tabular and (b-j) NN value representations. The black dashed lines indicate the performance of the optimal policy The green and magenta dashed line in (a) indicates the point after which DT and B planning's models become, and remain as, PXMs, respectively. Shaded regions are one standard error over (a) 250 and (b-j) 100 runs.

performance of DT planning degrades slightly with the increase in CMEs, this can indeed be easily mitigated by decreasing the search budget. However, as can be seen, even without any CMEs, just the usage of simulated experience alone can indeed slowdown, or even prevent (as in B(S)), B planning in reaching optimal (or close to optimal) performance, especially in hard-to-model environments such as SCS9N1 & LCS9N1.

TL Experiments. In Sec. 4.2, we first argued that B planning would again suffer more in reaching optimal (or close to optimal) performance on the training tasks in both TL settings. Then, we hypothesized that B planning would also suffer more in adapting to the subsequent test tasks in the first TL setting. Finally, we hypothesized that, under certain assumptions, DT planning would perform better than B planning on the test tasks in the second TL setting. In order to test the first two of these hypotheses, we compared the MIs of the two planning styles on a sequential version of the RDS environment [25] (see App. G.2 for the details). Results in Fig. 4f show that, similar to the P&L setting, the increasing usage of simulated experience can indeed slowdown, or even prevent (as in B(S)), B planning in reaching optimal (or close to optimal) performance on the training tasks, and that B planning indeed suffers more in adapting to the test tasks. And, in order to test the first and third hypotheses, we compared the two planning styles on the exact RDS environment. Results are shown in Fig. 4g-4j. As can be seen, the increasing usage of simulated experience can again slowdown, or even prevent (as in B(S)), B planning in reaching optimal (or close to optimal) performance on the training tasks. We can also see that DT planning indeed achieves significantly better performance across all test tasks with varying difficulties.

6 Conclusion and Discussion

To summarize, we analyzed DT and B planning algorithms in a unified way through the lens of dynamic programming. Overall, our findings suggest that even though DT planning does not perform as well as B planning in their CIs, due to the reasons detailed in this study, the MIs of it can perform on par or better than their B planning counterparts in both the P&L and TL settings. In this study, our main goal was to *understand* under what conditions and in which settings one planning style will perform better than the other in fast-response-requiring domains, and not to provide any practical insights at the moment. However, we believe that both the proposed unifying framework and our theoretical and empirical results can be helpful to the community in improving the two planning styles in potentially interesting ways. Also note that we were only interested in comparing the two planning styles in terms of the expected discounted return of their output policies. Though not the main focus of this study, other possible interesting comparison directions include comparing in terms of sample efficiency and real-time performance. Lastly, due to their easy-to-learn optimal policies and their suitability in designing controlled experiments, we have mainly performed our MI experiments on MG environments. However, experiments in other environments can be helpful in further validating the hypotheses of this study. We hope to tackle this in future work.

REFERENCES

[1] Ankesh Anand, Jacob C Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Theophane Weber, and Jessica B Hamrick. Procedural generalization by planning with self-supervised world models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=FmBegXJToY.

[2] Dimitri P Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021.

[3] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[4] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[5] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[6] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

[7] Jessica B Hamrick, Abram L. Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Holger Buesing, Petar Veličković, and Theophane Weber. On the role of planning in model-based deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=IrM64DGB21.

[8] Taher Jafferjee, Ehsan Imani, Erin Talvitie, Martha White, and Micheal Bowling. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*, 2020.

[9] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1181–1189. Citeseer, 2015.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

[11] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[12] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[14] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[15] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pp. 216–224. Elsevier, 1990.

[16] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] Erik Talvitie. Model regularization for stable sample rollouts. In *UAI*, pp. 780–789, 2014.

[19] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[20] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[21] Gerald Tesauro and Gregory Galperin. On-line policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, 9:1068–1074, 1996.

[22] Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32:14322–14333, 2019.

[23] Harm Van Seijen, Hadi Nekoei, Evan Racah, and Sarath Chandar. The loca regret: A consistent metric to evaluate model-based behavior in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6562–6572, 2020.

[24] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[25] Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`.

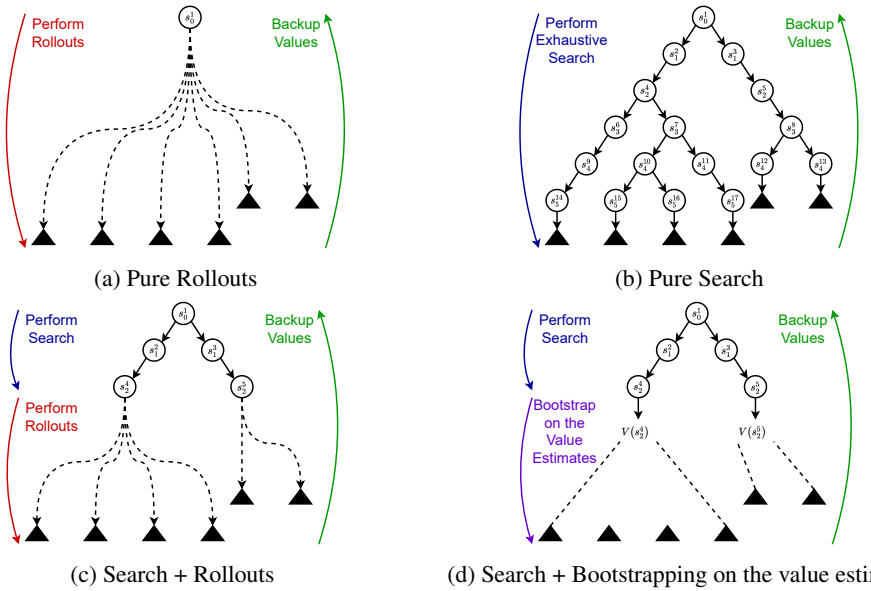
A PSEUDOCODES OF THE CIS OF THE TWO PLANNING STYLES

Algorithm 1 Tabular Online Monte-Carlo Planning (OMCP) [21] with an Adaptable Model

Initialize $\pi^i \in \mathbb{\Pi}$ as a random policy
Initialize $\bar{m}(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
$n_r \leftarrow$ number of episodes to perform rollouts
while $\bar{m}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy(MC_rollout$(S, \bar{m}, n_r, \pi^i)$)
        R, S', done $\leftarrow$ environment(A)
        Update $\bar{m}(S, A)$ with R, S', done
        $S \leftarrow S'$
    end while
end while
Return $\bar{m}(s,a)$

Algorithm 2 Tabular Exhaustive Search [4] with an Adaptable Model

Initialize $\bar{m}(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
$h \leftarrow$ search heuristic
while $\bar{m}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy(exhaustive_tree_search$(S, \bar{m}, h)$)
        R, S', done $\leftarrow$ environment(A)
        Update $\bar{m}(S, A)$ with R, S', done
        $S \leftarrow S'$
    end while
end while
Return $\bar{m}(s,a)$



(a) Pure Rollouts

(b) Pure Search

(c) Search + Rollouts

(d) Search + Bootstrapping on the value estimates

Figure 5: Various planning styles within DT planning in which planning is performed (i) by purely performing rollouts, (ii) by purely performing search, (iii) by performing rollouts after performing some amount of search, and (iv) by bootstrapping on the value estimates after performing some amount of search. The subscripts and the superscripts on the states indicate the time steps and state identifiers, respectively. The black traingles indicate the terminal states.

Algorithm 3 General Tabular Dyna-Q [15; 16]

Initialize $Q(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize $\bar{\bar{m}}(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
$\mathcal{SA}_{\text{prev}} \leftarrow \{\}$
$n_p \leftarrow$ number of time steps to perform planning
while Q and $\bar{\bar{m}}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy$(Q(S,\cdot))$
        $R, S', \text{done} \leftarrow \text{environment}(A)$
        $\mathcal{SA}_{\text{prev}} \leftarrow \mathcal{SA}_{\text{prev}} + \{(S,A)\}$
        Update $Q(S,A)$ with R, S', done
        Update $\bar{\bar{m}}(S,A)$ with R, S', done
        $i \leftarrow 0$
        while $i < n_p$ **do**
            $S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}} \leftarrow$ sample from $\mathcal{SA}_{\text{prev}}$
            $R_{\bar{\bar{m}}}, S'_{\bar{\bar{m}}}, \text{done}_{\bar{\bar{m}}} \leftarrow \bar{\bar{m}}(S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}})$
            Update $Q(S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}})$ with $R_{\bar{\bar{m}}}$, $S'_{\bar{\bar{m}}}$, $\text{done}_{\bar{\bar{m}}}$
            $i \leftarrow i + 1$
        end while
        $S \leftarrow S'$
    end while
end while
Return $Q(s,a)$

Algorithm 4 Tabular Dyna-Q of Interest

Initialize $Q(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize $\bar{\bar{m}}(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
while Q and $\bar{\bar{m}}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy$(Q(S,\cdot))$
        $R, S', \text{done} \leftarrow \text{environment}(A)$
        Update $\bar{\bar{m}}(S,A)$ with R, S', done
        $S \leftarrow S'$
    end while
    while Q has not converged **do**
        $S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}} \leftarrow$ sample from $\mathcal{S} \times \mathcal{A}$
        $R_{\bar{\bar{m}}}, S'_{\bar{\bar{m}}}, \text{done}_{\bar{\bar{m}}} \leftarrow \bar{\bar{m}}(S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}})$
        Update $Q(S_{\bar{\bar{m}}}, A_{\bar{\bar{m}}})$ with $R_{\bar{\bar{m}}}$, $S'_{\bar{\bar{m}}}$, $\text{done}_{\bar{\bar{m}}}$
    end while
end while
Return $Q(s,a)$

B Discussion on the Choice of the CIs of the Two Planning Styles

As indicated in the main paper, for DT planning, we study the OMCP algorithm of Tesauro & Galperin [21], and, for B planning, we study the Dyna-Q algorithm of Sutton [15; 16]. We choose these algorithms as they are easy to analyze. In this study, as we are interested in scenarios where the model has to be learned from pure interaction, we consider a version of the OCMP algorithm in which a parametric model is learned from experience (see Alg. 1 for the pseudocode). Note that this is the only difference compared to the original version of the OMCP algorithm. And, in order to make a fair comparison with this version of the OMCP algorithm, we consider a simplified version of the Dyna-Q algorithm (see Alg. 3 & 4 for the pseudocodes of the original and simplified versions, respectively). Compared to the original version of Dyna-Q, in this version, there are several minor differences:

- While planning, the agent can now sample states and actions that it has not observed or taken before. Note that this is also the case for the version of the OMCP algorithm considered in this study.
- Now, instead of using samples from both the environment and model, the agent updates its value estimator with samples only from the model. Note that the version of the OMCP algorithm considered in this study also makes use of only the model while performing planning.
- Now, instead of planning for a fixed number of time steps, the agent performs planning until its value estimator converges. Note that, in order to allow for sample efficiency, usually n_p is also set to high values in the original version of the Dyna-Q algorithm. Also note that, in order to properly evaluate the base policy, usually n_r is also set to high values in the version of the OMCP algorithm considered in this study. Thus, in practice, both the original version of the Dyna-Q algorithm and the version of the OMCP algorithm considered in this study also devote a significant budget to perform planning.
- Lastly, instead of performing planning after every time step, now the agent only performs planning after every episode. Rather than to allow for a fair comparison, this is to allow the Dyna-Q version of interest to be able operate in fast-response-requiring domains (planning until convergence after every time step would obviously slow down the response time of the algorithm and prevent it from operating in fast-response-requiring domains).

C PROOFS

Proposition 1. *Let $m \in \mathcal{M}$ be a PCM of m^* w.r.t. $\Pi = \{\pi^r_m, \pi^{ce}_m\} \subseteq \mathbb{\Pi}$ and J. Then, $J^{\pi^r_m}_{m^*} \geq J^{\pi^{ce}_m}_{m^*}$.*

Proof. This result directly follows from Defn. 1 & 5. Recall that, according to Defn. 5, given a $\pi^b \in \mathbb{\Pi}$, π^r_m and π^{ce}_m are the policies that are obtained after performing one-step PI and full PI on top of a π^b in model m, respectively. Thus, we have $J^{\pi^r_m}_m \leq J^{\pi^{ce}_m}_m$ [3], which, by Defn. 1, implies $J^{\pi^r_m}_{m^*} \geq J^{\pi^{ce}_m}_{m^*}$. □

Proposition 2. *Let $m \in \mathcal{M}$ be a PRM of m^* w.r.t. $\Pi = \{\pi^r_m, \pi^{ce}_m\} \subseteq \mathbb{\Pi}$ and J. Then, $J^{\pi^{ce}_m}_{m^*} \geq J^{\pi^r_m}_{m^*}$.*

Proof. This result directly follows from Defn. 2 & 5. Recall that, according to Defn. 5, given a $\pi^b \in \mathbb{\Pi}$, π^r_m and π^{ce}_m are the policies that are obtained after performing one-step PI and full PI on top of a π^b in model m, respectively. Thus, we have $J^{\pi^r_m}_m \leq J^{\pi^{ce}_m}_m$ [3], which, by Defn. 2, implies $J^{\pi^{ce}_m}_{m^*} \geq J^{\pi^r_m}_{m^*}$. □

Proposition 3. *Let $\bar{m} \in \mathcal{M}$ be a PCM or a PRM of m^* w.r.t. $\bar{\Pi} = \{\pi^r_{\bar{m}}, \pi^{ce}_{\bar{m}}\} \subseteq \mathbb{\Pi}$ and J, and let $\bar{\bar{m}} \in \mathcal{M}$ be a PNM of m^* w.r.t. $\bar{\bar{\Pi}} = \{\pi^r_{\bar{\bar{m}}}, \pi^{ce}_{\bar{\bar{m}}}\} \subseteq \mathbb{\Pi}$ and J. Then, $J^{\pi^r_{\bar{m}}}_{m^*} \geq J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*}$.*

Proof. This result directly follows from Defn. 3 & 5. Recall that, according to Defn. 5, given a $\pi^b \in \mathbb{\Pi}$, $\pi^{ce}_{\bar{\bar{m}}}$ is the policy that is obtained after performing full PI on top of π^b in model $\bar{\bar{m}}$. Thus, $\pi^{ce}_{\bar{\bar{m}}}$ is one of the optimal policies of model $\bar{\bar{m}}$ [3], which, by Defn. 3, implies $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} = \min_{\pi \in \Pi} J^{\pi}_{m^*}$ and thus $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} \leq J^{\pi}_{m^*} \forall \pi \in \mathbb{\Pi}$. This in turn implies $J^{\pi^r_{\bar{m}}}_{m^*} \geq J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*}$. □

Proposition 4. *Let $\bar{m} \in \mathcal{M}$ be a PCM or a PRM of m^* w.r.t. $\bar{\Pi} = \{\pi^r_{\bar{m}}, \pi^{ce}_{\bar{m}}\} \subseteq \mathbb{\Pi}$ and J, and let $\bar{\bar{m}} \in \mathcal{M}$ be a PXM of m^* w.r.t. $\bar{\bar{\Pi}} = \{\pi^r_{\bar{\bar{m}}}, \pi^{ce}_{\bar{\bar{m}}}\} \subseteq \mathbb{\Pi}$ and J. Then, $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} \geq J^{\pi^r_{\bar{m}}}_{m^*}$.*

Proof. This result directly follows from Defn. 4 & 5. Recall that, according to Defn. 5, given a $\pi^b \in \mathbb{\Pi}$, $\pi^{ce}_{\bar{\bar{m}}}$ is the policy that is obtained after performing full PI on top of π^b in model $\bar{\bar{m}}$. Thus, $\pi^{ce}_{\bar{\bar{m}}}$ is one of the optimal policies of model $\bar{\bar{m}}$ [3], which, by Defn. 4, implies $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} = \max_{\pi \in \Pi} J^{\pi}_{m^*}$ and thus $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} \geq J^{\pi}_{m^*} \forall \pi \in \mathbb{\Pi}$. This in turn implies $J^{\pi^{ce}_{\bar{\bar{m}}}}_{m^*} \geq J^{\pi^r_{\bar{m}}}_{m^*}$. □

D PSEUDOCODES OF THE MIS OF THE TWO PLANNING STYLES

Algorithm 5 The DT Planning Algorithm in Zhao et al. [25]

Initialize the parameters θ, η & ω of $\phi_\theta : \mathcal{O}_E \to \mathcal{S}_A$, $Q_\eta : \mathcal{S}_A \times \mathcal{A}_E \to \mathbb{R}$ & $\bar{m}_{p\omega} = (p_\omega, r_\omega, d_\omega)$
Initialize the replay buffer $\bar{m}_{np} \leftarrow \{\}$
$N_{ple} \leftarrow$ number of episodes to perform planning and learning
$N_{rbt} \leftarrow$ number of samples that the replay buffer must hold to perform planning and learning
$n_s \leftarrow$ number of time steps to perform search
$n_{bs} \leftarrow$ number of samples to sample from $\bar{m}_{np}$
$h \leftarrow$ search heuristic
$S \leftarrow$ replay buffer sampling strategy
$i \leftarrow 0$
while $i < N_{ple}$ **do**
    $O \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy(tree_search_with_bootstrapping$(\phi_\theta(O), \bar{m}_{p\omega}, Q_\eta, n_s, h)$)
        R, O', done $\leftarrow$ environment(A)
        $\bar{m}_{np} \leftarrow \bar{m}_{np} + \{(O, A, R, O', \text{done})\}$
        if $|\bar{m}_{np}| \geq N_{rbt}$ **then**
            $\mathcal{B}_{np} \leftarrow$ sample_batch$(\bar{m}_{np}, n_{bs}, S)$
            Update ϕ_θ, Q_η & $\bar{m}_{p\omega}$ with $\mathcal{B}_{np}$
        end if
        $O \leftarrow O'$
    end while
    $i \leftarrow i + 1$
end while
Return ϕ_θ, Q_η & $\bar{m}_{p\omega}$

Algorithm 6 The B Planning Algorithm in Zhao et al. [25]

Initialize the parameters θ, η & ω of $\phi_\theta : \mathcal{O}_E \to \mathcal{S}_A$, $Q_\eta : \mathcal{S}_A \times \mathcal{A}_E \to \mathbb{R}$ & $\bar{\bar{m}}_{p\omega} = (p_\omega, r_\omega, d_\omega)$
Initialize the replay buffer $\bar{\bar{m}}_{np} \leftarrow \{\}$ and the imagined replay buffer $\bar{\bar{m}}_{inp} \leftarrow \{\}$
$N_{ple} \leftarrow$ number of episodes to perform planning and learning
$N_{rbt} \leftarrow$ number of samples that the replay buffer must hold to perform planning and learning
$n_{ibs} \leftarrow$ number of samples to sample from $\bar{\bar{m}}_{inp}$
$n_{bs} \leftarrow$ number of samples to sample from $\bar{\bar{m}}_{np}$
$S \leftarrow$ replay buffer sampling strategy
$i \leftarrow 0$
while $i < N_{ple}$ **do**
    $O \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy$(Q_\eta(\phi_\theta(O), \cdot))$
        R, O', done $\leftarrow$ environment(A)
        $\bar{\bar{m}}_{np} \leftarrow \bar{\bar{m}}_{np} + \{(O, A, R, O', \text{done})\}$
        $\bar{\bar{m}}_{inp} \leftarrow \bar{\bar{m}}_{inp} + \{(\phi_\theta(O), A)\}$
        if $|\bar{\bar{m}}_{np}| \geq N_{rbt}$ **then**
            $\mathcal{B}_{inp} \leftarrow$ sample_batch$(\bar{\bar{m}}_{inp}, n_{ibs}, S)$
            $\mathcal{B}_p \leftarrow \mathcal{B}_{inp} + \bar{\bar{m}}_{p\omega}(\mathcal{B}_{inp})$
            $\mathcal{B}_{np} \leftarrow$ sample_batch$(\bar{\bar{m}}_{np}, n_{bs}, S)$
            Update ϕ_θ & Q_η with $\mathcal{B}_{np} + \mathcal{B}_p$
            Update ϕ_θ & $\bar{\bar{m}}_{p\omega}$ with $\mathcal{B}_{np}$
        end if
        $O \leftarrow O'$
    end while
    $i \leftarrow i + 1$
end while
Return ϕ_θ & Q_η

E Discussion on the Choice of the MIs of the Two Planning Styles

As indicated in the main paper, we study the DT and B planning algorithms in Zhao et al. [25]. More specifically, for DT planning, we study the "UP" algorithm, and, for B planning, we study the "Dyna" algorithm in [25]. We choose these algorithms as they are reflective of many of the properties of their state-of-the-art (SOTA) counterparts (MuZero [12] and SimPLe [26] / DreamerV2 [6], respectively) and their code is publicly available[2]. The pseudocodes of these algorithms are presented in Alg. 5 & 6, respectively. Note that, similar to their SOTA counterparts, these two algorithms do not employ the "bottleneck mechanism" introduced in [25]. Some of the important similarities and differences between these algorithms and their SOTA counterparts, are as follows:

1. Similarities and differences between the DT planning algorithm in Zhao et al. [25] and MuZero [12]
    - Similar to MuZero, the DT planning algorithm in [25] also performs planning with both a parametric and non-parametric model.
    - Similar to MuZero, the DT planning algorithm in [25] also represents its parametric model using NNs.
    - Similar to MuZero, the DT planning algorithm in [25] also learns a parametric model through pure interaction with the environment. However, rather than unrolling the model for several time steps and training it with the sum of the policy, value and reward losses as in MuZero, it unrolls the model for only a single time step and trains it with the sum of the value, dynamics, reward and termination losses (see the total loss term in Sec. 3 of [25]).
    - Lastly, similar to MuZero, the DT planning algorithm in [25] selects actions by directly bootstrapping on the value estimates of a continually improving policy (without performing any rollouts), which is obtained by planning with a non-parametric model, after performing some amount search with a parametric model. However, rather than performing the search using Monte-Carlo Tree Search (MCTS, [10]) as in MuZero, it uses best-first search (during training) and random search (during evaluation) (see App. H of [25] for the details of the search procedures).
2. Similarities and differences between the B planning algorithm in Zhao et al. [25] and SimPLe [26] / DreamerV2 [6]
    - Similar to SimPLe / DreamerV2, the B planning algorithm in [25] also performs planning with a parametric model. Additionally, it also performs planning with a non-parametric model.
    - Similar to SimPLe / DreamerV2, the B planning algorithm in [25] also represents its parametric model using NNs and it also updates its value estimator using the simulated data generated with this model.
    - Similar to SimPLe / DreamerV2, the B planning algorithm in [25] also learns a parametric model through pure interaction with the environment. However, rather than performing planning with this model after allowing for an initial kickstarting period as in SimPLe / DreamerV2 (referred to as a "world model" learning period), for a fair comparison with the DT planning algorithm, it starts to perform planning right at the beginning of the model learning process.
    - Lastly, similar to SimPLe / DreamerV2, the B planning algorithm in [25] selects actions by simply querying its value estimator.

Even though there are some differences between the planning algorithms in [25] and their SOTA counterparts, except for the kickstarting period in SOTA B planning algorithms, these are just minor implementation details that would not have any impact on the conclusions of this study. Although the kickstarting period can mitigate the harmful simulated data problem to some degree (or even prevent it if the period is sufficiently long), allowing for it would definitely prevent a fair comparison with the DT planning algorithm in [25]. This is why we did not allow for it in our experiments.

[2] `https://github.com/mila-iqia/Conscious-Planning`

F Discussion on the Combined View of Models

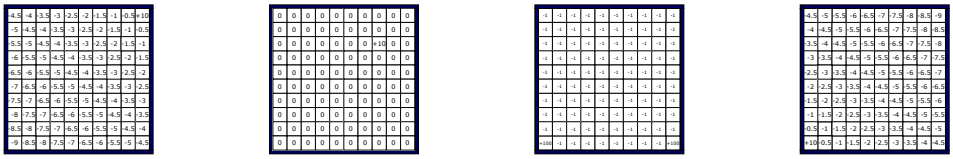
In order to be able to view the DT and B planning algorithms that perform planning with both a parametric and non-parametric model through our proposed dynamic programming framework, we view the two separate models of these algorithms as a single combined model. This becomes obvious for B planning algorithms if one notes that they perform planning with a batch of data that is jointly generated by both a parametric and non-parametric model (see e.g., line 20 in Alg. 6 in which ϕ_θ and Q_η are updated with batch of data that is jointly generated by both $\bar{\bar{m}}_{p\omega}$ and $\bar{\bar{m}}_{np}$), which can be thought of performing planning with a batch of data that is generated by a single combined model. It also becomes obvious for DT planning algorithms if one notes that they perform planning by first performing search with a parametric model, and then by bootstrapping on the value estimates of a continually improving policy that is obtained by planning with a non-parametric model (see e.g., line 13 in Alg. 5 in which action selection is done with both $\bar{m}_{p\omega}$ and Q_η (which is obtained by planning with $\bar{m}_{np}$)), which can be thought of performing planning with a single combined model that is obtained by concatenating the parametric and non-parametric models.

G Experimental Details

In this section, we provide the details of the experiments that are performed in Sec. 5. This also includes the implementation details of the CIs and MIs of the two planning styles considered in this study.

G.1 Details of the CI Experiments

In all of the CI experiments, we have calculated the performance with a discount factor (γ) of 0.9.



(a) SG Rewards (b) m_8 Rewards (c) Initial PDM Rewards (d) Test Task Rewards

Figure 6: Reward functions of (a) the SG environment, (b) the m_8 model, (c) the initial PDM, and (d) the test task.

Environment & Models. All of the experiments in Sec. 5.1 are performed on the SG environment. Here, the agent spawns in state S and has to navigate to the goal state depicted by G. At each time step, the agent receives an (x, y) pair, indicating its position, and based on this, selects an action that moves it to one of the four neighboring cells with a slip probability of 0.05. The agent receives a negative reward that is linearly proportional to its distance from G and a reward of $+10$ if it reaches G (see Fig. 6a). The agent-environment interaction lasts for a maximum of 100 time steps, after which the episode terminates with a reward of 0 if the agent was not able to reach the goal state G. (**PP Setting**) For the experiments in the PP setting, the agent is provided with series of models in which it receives a reward of $+10$ if it reaches the goal state and a reward of 0 elsewhere. For example, see the reward function of model m_8 in Fig. 6b. Note that these models have the same transition distribution and initial state distribution with the SG environment. (**P&L Setting**) For the experiments in the P&L setting, the models of both of the planning styles are initialized as a hand-designed PDM with a reward function as in Fig. 6c and with a goal state located at the bottom right corner. Note that in these experiments, we have assumed that the agent already has access to the transition distribution and initial state distribution of the environment, and only has to learn the reward function. (**TL Setting**) Finally, for the experiments in the TL setting, we considered a test task with a reward function as in Fig. 6d, which is a transposed version of the training task's reward function (see Fig. 6a). Note again that we have assumed that the agent already has access to the transition distribution and initial state distribution of the environment, and only has to learn the reward function.

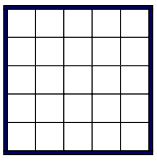


Figure 7: The form of SA used in this study.

Implementation Details of the CIs. For our CI experiments, we considered specific versions of the OMCP algorithm of Tesauro & Galperin [21] and the Dyna-Q algorithm of Sutton [15; 16]. The pseudocodes of these algorithms are presented in Alg. 1 & 4, respectively, and the details of them are provided in Table 1 & 2, respectively. Note that in Alg. 1, n_r (the number of episodes to perform rollouts) is set to a high value so that the input policy π^i can properly be evaluated.

Table 1: Details and hyperparameters of Alg. 1.

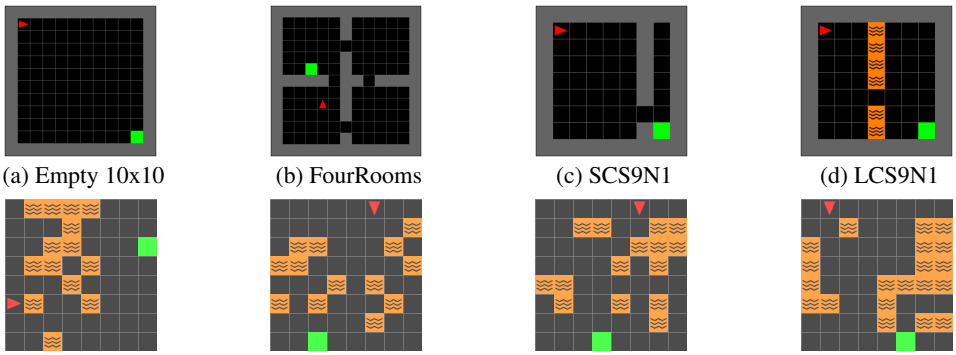
π^i	a deterministic random policy
$\bar{m}$	a tabular model (initialized as a hand-designed PDM (see Fig. 6c))
n_r	50
ϵ	linearly decays from 1.0 to 0.0 over the first 20 episodes
SA (in function approximation experiments)	an aggregation of the form in Fig. 7

Table 2: Details and hyperparameters of Alg. 4.

Q	a tabular value function (initialized as zero $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$)
$\bar{\bar{m}}$	a tabular model (initialized as a hand-designed PDM (see Fig. 6c))
ϵ	linearly decays from 1.0 to 0.0 over the first 20 episodes
SA (in function approximation experiments)	an aggregation of the form in Fig. 7

G.2 Details of the MI Experiments

In all of the MI experiments on the SG environment, we have calculated the performance with a discount factor (γ) of 0.9, and in all of the MI experiments on the MG environments, we have calculated the performance with a discount factor of 0.99.



(a) Empty 10x10 (b) FourRooms (c) SCS9N1 (d) LCS9N1

(e) RDS Train (0.35) (f) RDS Test (0.25) (g) RDS Test (0.35) (h) RDS Test (0.45)

Figure 8: (a-d) Several environments that either pre-exist in or are manually built on top of MG. (e-h) The training tasks and test tasks of varying difficulty in the RDS environment [25]. Note that the test tasks are just transposed versions of the training tasks.

Environments & Models. In Sec. 5.2, part of our experiments were performed both on the SG environment and part of them were performed on MG environments. The details of these environments and their corresponding models are as follows:

- **SG Environment.** To learn about the SG environment, we refer the reader to Sec. G.1 as we have used the same environment in the MI experiments as well. (**P&L Setting**) To learn about the models of both planning styles, we also refer the reader to the P&L Setting of Sec. G.1 as have used the same models in the MI experiments as well.
- **MG Environments.** In the MG environments, the agent, depicted in red, has to navigate to the green goal cell while avoiding the orange lava cells (if there are any). At each time step, the agent receives a grid-based observation that contains its own position and the positions of the goal, wall and lava cells, and based on this, selects an action that either turns it left or right, or steps it forward. If the agent steps on a lava cell, the episode terminates with no reward, and if it reaches the goal cell, the episode terminates with a reward of $+1$.[3] More

[3]Note that this is not the original reward function of MG environments. In the original version, the agent receives a reward of $+1 - 0.9(t/T)$, where t is the number of time steps taken to reach the goal cell and T is

on the details of these environments can be found in Chevalier-Boisvert et al. [5]. (**P&L Setting**) For the P&L setting, we performed experiments on the Empty 10x10, FourRooms, SimpleCrossingS9N1 (SCS9N1) and LavaCrossingS9N1 (LCS9N1) environments (see Fig. 8a, 8b, 8c, & 8d, respectively). While the last two of these environments already pre-exist in MG, the first two of them are manually built environments. Specifically, the Empty 10x10 environment is obtained by expanding the Empty 8x8 environment and the 10x10 FourRooms environment is obtained by contracting the 16x16 FourRooms environment in Chevalier-Boisvert et al. [5]. (**TL Setting**) For the TL setting, we performed experiments on the sequential and regular versions of the RDS environment considered in [25] (see Fig. 8e-8h).[4] In the sequential version, which we call RDS Sequential, the agent is trained on training tasks with difficulty 0.35 (see Fig. 8e) and then it is allowed to adapt to subsequent test tasks (a transposed version of the training tasks) with difficulty 0.35 (see Fig. 8g). In the regular version (the version considered in [25]), the agent is trained on training tasks with difficulty 0.35 (see Fig. 8e) and during the training process it is periodically evaluated on test tasks with difficulties varying from 0.25 to 0.45 (see Fig. 8f-8h). Note that the difficulty parameter here controls the density of the lava cells. Also note that with every reset of the episode, a new lava cell pattern is (procedurally) generated for both the training tasks and test tasks. More on the details of the RDS environment can be found in [25].

Finally, note that, as opposed to the experiments on SG environment, for both the P&L and TL settings, we did not enforce any kind of structure on the models of the agent, and just initialized them randomly. We also note that, in our experiments with RDS Sequential, we reinitialized the non-parametric models (replay buffers) of both planning styles after the tasks switch from the training tasks to the test tasks.

Implementation Details of the MIs. For our MI experiments, we considered the DT and B planning algorithms in Zhao et al. [25] (see Sec. E), whose pseudocodes are presented in Alg. 5 & 6, respectively. The details of these algorithms are provided in Table 3 & 4, respectively. For more details (such as the NN architectures, replay buffer sizes, learning rates, exact details of the tree search, ...), we refer the reader to the publicly available code and the supplementary material of [25].

Table 3: Details and hyperparameters of Alg. 5.

ϕ_θ	MiniGrid bag of words feature extractor
Q_η	regular NN (P&L setting), NN with attention (for set-based representations) (TL setting)
$\bar{m}_{p\omega}$	regular NN (P&L setting), NN with attention (for set-based representations) (TL setting) (bottleneck mechanism is disabled for both of the setting)
N_{ple}	50M
N_{rbt}	50k
n_s	1 (DT(1)), 5 (DT(5)), 15 (DT(15))
n_{bs}	128 (P&L setting), 64 (TL setting)
h	best-first search (training), random search (evaluation)
S	random sampling
ϵ	linearly decays from 1.0 to 0.0 over the first 1M time steps

Table 4: Details and hyperparameters of Alg. 6.

ϕ_θ	MiniGrid bag of words feature extractor
Q_η	regular NN (P&L setting), NN with attention (for set-based representations) (TL setting)
$\bar{\bar{m}}_{p\omega}$	regular NN (P&L setting), NN with attention (for set-based representations) (TL setting) (bottleneck mechanism is disabled for both of the settings)
N_{ple}	50M
N_{rbt}	50k
(n_{ibs}, n_{bs})	$(0, 128)$ (B(R)), $(128, 128)$ (B(R+S)), $(128, 0)$ (B(S)) (P&L setting), $(0, 64)$ (B(R)), $(64, 64)$ (B(R+S)), $(64, 0)$ (B(S)) (TL setting)
S	random sampling
ϵ	linearly decays from 1.0 to 0.0 over the first 1M time steps

the maximum episode length, if it reaches the goal. We modified the reward function in order to obtain more intuitive results.

[4]Note that the RDS environment is an environment that is built on top of MG.

Note that the publicly available code of [25] only contains B planning algorithms in which the model is a model over the observations (and not the states), which for some reason causes the B planning algorithm of interest to perform very poorly (see the plots in [25]). For this reason and in order to make a fair comparison with DT planning, we have implemented a version of the B planning algorithm in which the model is a model over the states and we performed all of our experiments with this version of the algorithm. Also note that, while we have used regular representations in our P&L experiments, in order to deal with the large number of tasks, we have made use of set-based representations in our TL experiments (see [25] for the details of this representation).

Additionally, we also performed experiments with simplified tabular versions of the MIs of the two planning styles, whose pseudocodes are presented in Alg. 7 & 8, respectively. The details of these algorithms are provided in Table 5 & 6, respectively.

Algorithm 7 Modernized Version of Tabular OMCP with both a Parametric and Non-Parametric Model

Initialize $Q(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize $\bar{m}_p(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize the replay buffer $\bar{m}_{np} \leftarrow \{\}$
$n_s \leftarrow$ number of time steps to perform search
$h \leftarrow$ search heuristic
while $\bar{m}_p$ and $\bar{m}_{np}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy(tree_search_with_bootstrapping$(S, \bar{m}_p, Q, n_s, h)$)
        R, S', done $\leftarrow$ environment(A)
        $\bar{m}_{np} \leftarrow \bar{m}_{np} + \{(S, A, R, S', \text{done})\}$
        $S_{\bar{m}_{np}}, A_{\bar{m}_{np}}, R_{\bar{m}_{np}}, S'_{\bar{m}_{np}}, \text{done}_{\bar{m}_{np}} \leftarrow$ sample from $\bar{m}_{np}$
        Update Q & $\bar{m}_p$ with $S_{\bar{m}_{np}}, A_{\bar{m}_{np}}, R_{\bar{m}_{np}}, S'_{\bar{m}_{np}}, \text{done}_{\bar{m}_{np}}$
        $S \leftarrow S'$
    end while
end while
Return Q & $\bar{m}_p(s,a)$

Table 5: Details and hyperparameters of Alg. 7.

Q	a tabular value function (initialized as zero $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$)
$\bar{m}_p$	a tabular model (initialized as a hand-designed PDM (see Fig. 6c))
n_s	$\lvert\mathcal{A}\rvert$
h	breadth-first search
ϵ	linearly decays from 1.0 to 0.0 over the first 20 episodes

Table 6: Details and hyperparameters of Alg. 8.

Q	a tabular value function (initialized as zero $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$)
$\bar{\bar{m}}_p$	a tabular parametric model (initialized as a hand-designed PDM (see Fig. 6c))
ϵ	linearly decays from 1.0 to 0.0 over the first 20 episodes

H Additional Results

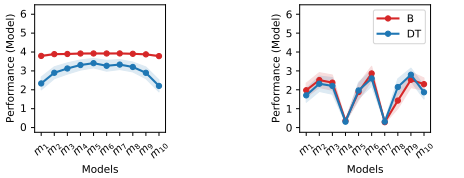
In this section, we provide complementary results to our empirical results in Sec. 5. Specifically, we provide (i) performance plots that are obtained by evaluating the different planning styles in their corresponding models and (ii) total reward plots that are obtained by evaluating the different planning styles in both the considered environments and in their corresponding models. Note that while the performance plots are obtained by the measure in (1), the total reward plots are obtained by simply adding the rewards obtained by the agent throughout the episodes (which is actually the expected *undiscounted* return, i.e., when we use $\gamma = 1.0$ in measure (1)).

Algorithm 8 Modernized Version of Tabular Dyna-Q of Interest with both a Parametric and Non-Parametric Model

Initialize $Q(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize $\bar{\bar{m}}_p(s,a)\ \forall s \in \mathcal{S}$ & $\forall a \in \mathcal{A}$
Initialize the replay buffer $\bar{\bar{m}}_{np} \leftarrow \{\}$
while Q, $\bar{\bar{m}}_p$ and $\bar{\bar{m}}_{np}$ has not converged **do**
    $S \leftarrow$ reset environment
    while not done **do**
        $A \leftarrow \epsilon$-greedy$(Q(S,\cdot))$
        R, S', done $\leftarrow$ environment(A)
        Update $\bar{\bar{m}}_p(S,A)$ with R, S', done
        $\bar{\bar{m}}_{np} \leftarrow \bar{\bar{m}}_{np} + \{(S,A,R,S',\text{done})\}$
        $S \leftarrow S'$
    end while
    while Q has not converged **do**
        $S_{\bar{\bar{m}}_p}, A_{\bar{\bar{m}}_p} \leftarrow$ sample from $\mathcal{S} \times \mathcal{A}$
        $R_{\bar{\bar{m}}_p}, S'_{\bar{\bar{m}}_p}, \text{done}_{\bar{\bar{m}}_p} \leftarrow \bar{\bar{m}}_p(S_{\bar{\bar{m}}_p}, A_{\bar{\bar{m}}_p})$
        Update $Q(S_{\bar{\bar{m}}_p}, A_{\bar{\bar{m}}_p})$ with $R_{\bar{\bar{m}}_p}$, $S'_{\bar{\bar{m}}_p}$, $\text{done}_{\bar{\bar{m}}_p}$
        $S_{\bar{\bar{m}}_{np}}, A_{\bar{\bar{m}}_{np}}, R_{\bar{\bar{m}}_{np}}, S'_{\bar{\bar{m}}_{np}}, \text{done}_{\bar{\bar{m}}_{np}} \leftarrow$ sample from $\bar{\bar{m}}_{np}$
        Update $Q(S_{\bar{\bar{m}}_{np}}, A_{\bar{\bar{m}}_{np}})$ with $R_{\bar{\bar{m}}_{np}}$, $S'_{\bar{\bar{m}}_{np}}$, $\text{done}_{\bar{\bar{m}}_{np}}$
    end while
end while
Return $Q(s,a)$

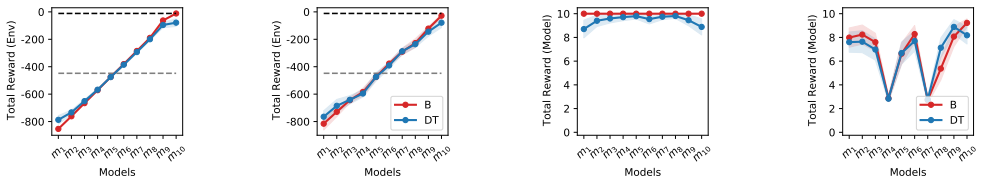
H.1 Experiments with CIs

H.1.1 PP Experiments



(a) Tabular (b) State Aggregarion

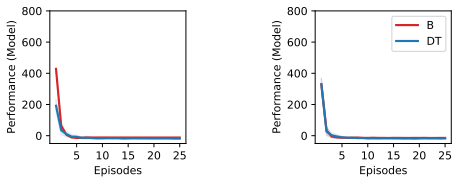
Figure 9: The performance of the CIs of DT and B planning in their corresponding models (learned on the SG environment) in the PP setting with tabular and SA value estimator representations. Shaded regions are one standard error over 250 runs.



(a) Tabular (b) State Aggregation (c) Tabular (d) State Aggregation

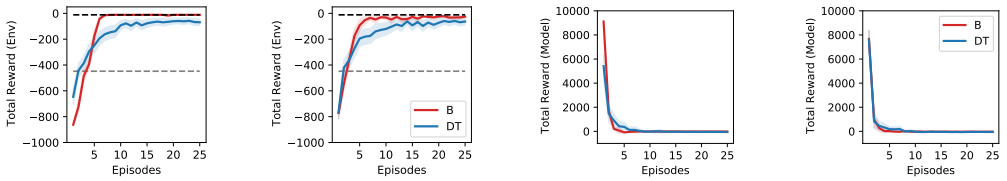
Figure 10: The total reward obtained by the CIs of DT and B planning (a, b) on the SG environment and (c, d) in their corresponding models (learned on the SG environment) in the PP setting with tabular and SA value estimator representations. Black & gray dashed lines indicate total reward obtained by the optimal & random policies, respectively. Shaded regions are one standard error over 250 runs.

H.1.2 P&L EXPERIMENTS



(a) Tabular (b) State Aggregation

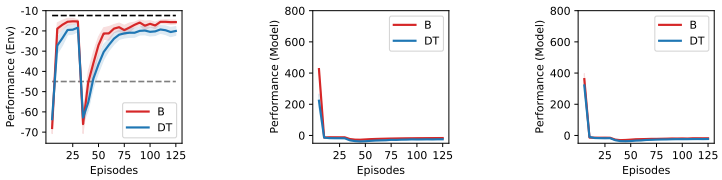
Figure 11: The performance of the CIs of DT and B planning in their corresponding models (learned on the SG environment) in the P&L setting with tabular and SA value estimator representations. Shaded regions are one standard error over 250 runs.



(a) Tabular (b) State Aggregation (c) Tabular (d) State Aggregation

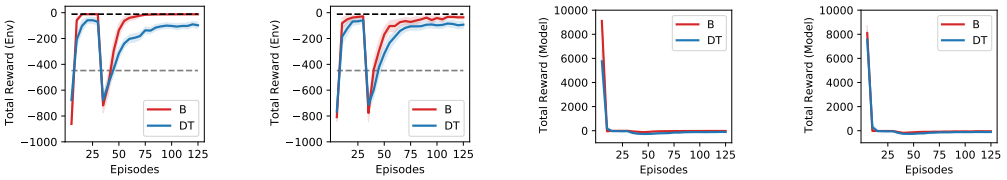
Figure 12: The total reward obtained by the CIs of DT and B planning (a, b) on the SG environment and (c, d) in their corresponding models (learned on the SG environment) in the P&L setting with tabular and SA value estimator representations. Black & gray dashed lines indicate total reward obtained by the optimal & random policies, respectively. Shaded regions are one standard error over 250 runs.

H.1.3 TL EXPERIMENTS



(a) State Aggregation (b) Tabular (c) State Aggregation

Figure 13: The performance of the CIs of DT and B planning (a) on the SG environment and (b, c) in their corresponding models (learned on the SG environment) in the TL setting with tabular and SA value estimator representations. Black & gray dashed lines indicate the performance of the optimal & random policies, respectively. Shaded regions are one standard error over 250 runs.

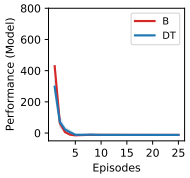


(a) Tabular (b) State Aggregation (c) Tabular (d) State Aggregation

Figure 14: The total reward obtained by the CIs of DT and B planning (a, b) on the SG environment and (c, d) in their corresponding models (learned on the SG environment) in the TL setting with tabular and SA value estimator representations. Black & gray dashed lines indicate total reward obtained by the optimal & random policies, respectively. Shaded regions are one standard error over 250 runs.

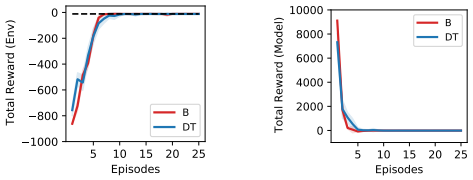
H.2 EXPERIMENTS WITH MIS

H.2.1 P&L EXPERIMENTS



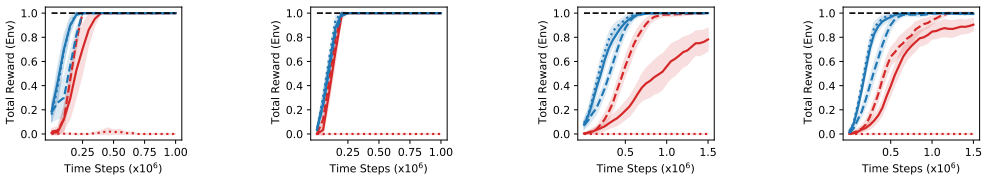
(a) Tabular

Figure 15: The performance of the MIs of DT and B planning in their corresponding models (learned on the SG environment) in the P&L setting with tabular value estimator representations. Shaded regions are one standard error over 250 runs.



(a) Tabular (b) Tabular

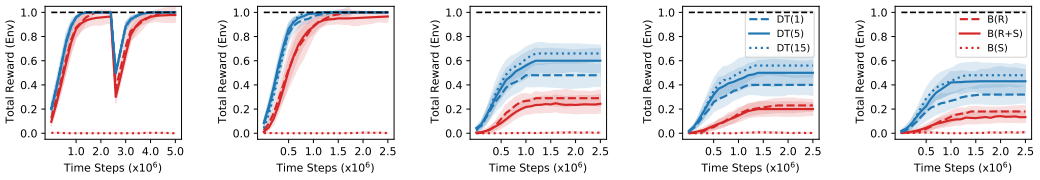
Figure 16: The total reward obtained by the MIs of DT and B planning (a) on the SG environment and (b) in their corresponding models (learned on the SG environment) in the P&L setting with tabular value estimator representations. The black dashed line indicates the total reward obtained by the optimal policy. Shaded regions are one standard error over 250 runs.



(a) Empty 10x10 (b) FourRooms (c) SCS9N1 (d) LCS9N1

Figure 17: The total reward obtained by the MIs of DT and B planning in the P&L setting with NN value estimator representations. The black dashed lines indicate the total reward obtained by the optimal policy in the corresponding environment. Shaded regions are one standard error over 100 runs.

H.2.2 TL EXPERIMENTS



(a) RDS Sequential (b) RDS Train (0.35) (c) RDS Test (0.25) (d) RDS Test (0.35) (e) RDS Test (0.45)

Figure 18: The total reward obtained by the MIs of DT and B planning in the TL setting with NN value estimator representations. The black dashed lines indicate the total reward obtained by the optimal policy in the corresponding environment. Shaded regions are one standard error over 100 runs.