# Bayesian Hierarchical Reinforcement Learning

**Anonymous Author(s)**
Affiliation
Address
`email`

## Abstract

We describe an approach to incorporating Bayesian priors in the MAXQ framework for hierarchical reinforcement learning (HRL). We define priors on the primitive environment model. Since models for composite tasks can be complex, we use a mixed model-based/model-free learning approach to find an optimal hierarchical policy. We show empirically that (i) our approach results in improved convergence over non-Bayesian baselines, given sensible priors, (ii) good priors on the environment model can, in some cases, reduce the convergence advantage of HRL over flat RL, and (iii) taking advantage of the structural decomposition induced by the task hierarchy reduces the computational cost of Bayesian reinforcement learning.

## 1 Introduction

Reinforcement learning (RL) is a well known framework that formalizes decision making in unknown, uncertain environments. RL agents learn policies that map environment states to available actions while optimizing some measure of long-term utility. While various algorithms have been developed for RL [17], and applied successfully to a variety of tasks [12], the standard RL setting suffers from at least two drawbacks. First, it is difficult to scale standard RL approaches to large state spaces (the well-known "curse of dimensionality"). Second, vanilla RL approaches do not incorporate prior knowledge about the environment and good policies.

*Hierarchical reinforcement learning* (HRL) [2] attempts to alleviate the scaling problem by simplifying the overall decision making problem in different ways. For example, we could introduce macro-operators that can be used by the agent as part of its policy. These operators carry out sequences of primitive actions, so planning at the level of these operators may be expected to result in simpler policies [15]. Another idea is to decompose the task's overall value function in a variety of ways, for example by defining task hierarchies [7] or partial programs with choice points [1]. Here, the structure of the decomposition provides several benefits: first, for the "higher level" subtasks, policies are defined by calling "lower level" subtasks (which may themselves be quite complex); as a result policies for higher level subtasks may be expressed compactly. Second, a task hierarchy or partial program can impose constraints on the space of policies by encoding knowledge about the structure of good policies and thereby reduce the search space. Third, learning within subtasks allows *state abstraction*, that is, some state variables can be ignored because they do not affect the policy within that subtask. This also simplifies the learning problem.

While HRL attempts to address the scalability issue, it still does not take into account any probabilistic prior knowledge the agent may have about the task it wishes to solve. For example, the agent may have some idea of the structure of the state space, where high/low utility states may be located and what their utilities may be, or even some idea about the approximate shape of the value function or policy. *Bayesian reinforcement learning* addresses this issue by incorporating priors on models [5], value functions [6, 8] or policies [10] into the learning process. Specifying good priors leads to many benefits, such as initial good policies, directed exploration towards regions of uncertainty, and faster convergence to the optimal policy.

In our work, we propose an approach that incorporates Bayesian priors into hierarchical reinforcement learning. We extend the MAXQ framework for HRL [7], that specifies a principled way of decomposing the overall task into subtasks so that value functions of the individual subtasks can be combined to recover the value function of the overall task. We extend this framework by incorporating priors on the primitive environment model. In order to avoid building models for composite tasks (which can be very complex), we adopt a mixed model-based/model-free learning approach. Our approach is integrated into the standard MAXQ model-free learning algorithm in a straightforward way. We empirically evaluate our algorithm to understand the effect of the priors in addition to the task hierarchy. Our experiments indicate that: (i) taking advantage of probabilistic prior knowledge can lead to faster convergence, even for HRL, (ii) however, in some cases, good priors on the environment model can reduce the convergence advantage of HRL over flat RL, and (iii) taking advantage of the task hierarchy can reduce the computational cost of Bayesian RL, which generally tends to be very computation-intensive. In this way Bayesian RL and HRL are synergistic: Bayesian RL can bring improved convergence to HRL, while HRL can reduce the cost of Bayesian RL.

We note that our work assumes the probabilistic priors to be given in advance and focuses on learning with them. Other work has addressed the issue of obtaining these priors. For example, one source of prior information is multi-task reinforcement learning [13, 19], where an agent solves a sequence of RL tasks, and uses the solutions of previous tasks to build priors over models or policies for future tasks. We also assume the task hierarchy is given. Other work has explored learning task hierarchies in different settings [14].

## 2 Background and Related Work

In this section, we briefly describe the MAXQ value function decomposition and standard Bayesian model-based reinforcement learning.

The MAXQ framework for HRL [7] decomposes the overall task into an acyclic task graph. The lowest level of the task graph represents primitive actions in the world. Other levels represent composite tasks, with the top level representing the whole task. Each composite subtask $T_i$ defines a semi-Markov Decision Process (SMDP) with parameters $\langle S_i, X_i, C_i, G_i \rangle$. $S_i$ is a predicate that defines the set of "non-terminal" states for $T_i$, where $T_i$ may be called by its parent. $G_i$ is a predicate that defines a set of "goal" states for $T_i$. Thus MAXQ tries to learn a value function (or policy) for $T_i$ that takes it from any $S_i$ to any $G_i$. The actions available within $T_i$ are described by the set of "child tasks" $C_i$. Finally, $X_i$ denotes the set of "relevant state variables" for $T_i$. These are the variables that parametrize the value function (or policy) that MAXQ learns for $T_i$. Often, we unify the predicates defining (the negation of) $S_i$ and $G_i$ into a single "termination" predicate, $P_i$. A (state, action, next-state) *(s,a,s')* triple where $P_i(s)$ is false, $P_i(s')$ is true, $a \in C_i$, and the transition probability $P(s'|s,a) > 0$ is called an *exit* of the subtask $T_i$.

A subtask can be invoked in any state $s$ where $P_i(s)$ is false, and it terminates when an exit causes $P_i(s')$ to become true. The set $S_i$ is defined using a projection function that maps a world state to an abstract state defined by a subset of the state variables. If the abstraction function is *safe*, it will only merge the world states that have the same value function into an abstracted state. A *local policy* for the subtask $T_i$ is a mapping from the states $S_i$ to $C_i$. A hierarchical policy $\pi$ for the overall task is an assignment of a local policy to each SMDP $T_i$. A *hierarchically optimal policy* for a given MAXQ graph is a hierarchical policy that has the best possible reward. A hierarchical policy is said to be *recursively optimal* if the local policy for each subtask is optimal given that all its subtask policies are optimal. Given a task graph, model-free [7] or model-based [11] methods can be used to learn value functions for each task-subtask pair, and used to determine the hierarchical policy (which subtask to call at each level for each state). In the model-free method, a policy is produced by maintaining a value and a *completion* function for each subtask. For a task $i$, the value $V(a, s)$ denotes the expected value of calling child task $a$ in state $s$. This is (recursively) estimated as the expected cumulative reward obtained while executing $a$. The completion function $C(i, s, a)$ denotes the expected cumulative reward obtained while *completing* $i$ after having called $a$ in $s$. The central idea behind MAXQ is that the value of $i$, $V(i, s)$, can be (recursively) decomposed in terms

**Algorithm 1** BAYESIAN_MAXQ

```
Input: Task $i$, State $s$, Update Interval $k$, Simulation Episodes $Sim$
Output: Next state $s'$, steps taken $N$
 1: if $i$ is primitive then
 2:     Execute $i$, observe $r$, $s'$
 3:     Update current posterior parameters $\Psi$ using ($s$, $i$,$r$, $s'$)
 4:     Update current value estimate: $V(i,s) \leftarrow (1-\alpha) \cdot V(i,s) + \alpha \cdot r$
 5:     $Count(i) \leftarrow Count(i) + 1$
 6:     return $(s', 1)$
 7: else
 8:     $N \leftarrow 0$ {$i$ is composite}
 9:     while $i$ is not terminated do
10:         RECOMPUTE_VALUE$(i, k, Sim)$
11:         $a \leftarrow \pi_g(i,s)$ {$\pi_g$ is the greedy policy from the current value function}
12:         $(s', N_a) \leftarrow$ BAYESIAN_MAXQ($a$, $s$) {recursive call to child task}
13:         $a^* \leftarrow \arg\max_{a'} \big[C(i,s',a') + V(a',s')\big]$ {update completion function in $i$}
14:         $C(i,s,a) \leftarrow (1-\alpha) \cdot C(i,s,a) + \alpha \cdot \gamma^{N_a} \big[C(i,s',a^*) + V(a^*,s')\big]$
15:         $s \leftarrow s'$
16:         $N \leftarrow N + N_a$
17:         $Count(i) \leftarrow Count(i) + 1$
18:     end while
19:     return $(s', N)$
20: end if
```

of $V(a,s)$ and $C(i,s,a)$. In fact, by maintaining and updating $C(i,s,a)$ and $V(a,s)$ for primitive $a$, it is possible to reconstruct $V(i,s)$ for any $i$ and so construct a hierarchical policy.

Bayesian reinforcement learning methods incorporate probabilistic prior knowledge on models [5], value functions [6, 8], policies [10] or combinations [9]. One basic Bayesian model-based RL algorithm proceeds as follows. At each step, a distribution over model parameters is maintained. Initially, this is just the prior distribution. At each step, a model is sampled from this distribution (Thompson sampling [18, 16]). This model is then solved and actions are taken according to the policy obtained. This yields observations about the environment model. These observations are used to update the parameters of the current distribution to create a posterior distribution over models. This procedure is then iterated. As more observations are obtained, the posterior distribution is refined to focus around the true environment model, and the policy asymptotically converges to the optimal policy for this model. Variations of this basic idea have also been investigated, for example, some work converts the distribution over models to an empirical distribution over $Q$-functions, and produces policies by sampling from this distribution instead [5].

Relatively little work exists that attempts to incorporate probabilistic priors into HRL. We have found one preliminary attempt [4]. This work builds on the RMAX+MAXQ [11] method, which extends RMAX [3] to MAXQ by maintaining models for each subtask and performing RMAX-style exploration in each. The Bayesian approach adds priors onto each subtask model and performs (separate) Bayesian model-based learning for each subtask. [1] In our approach, we do not construct models for subtasks, which can be very complex in general. Instead, we only maintain distributions over primitive actions, and use a mixed model-based/model-free learning algorithm that naturally integrates into the standard MAXQ learning algorithm.

## 3 Bayesian MAXQ Algorithm

In this section, we describe our approach to incorporating probabilistic priors into HRL. The intuition behind our approach is quite straightforward: we follow the basic Bayesian model-based RL procedure outlined in Section 2 and adapt it to the MAXQ framework. At each step we have a distribution over environment models (initially the prior). Our algorithm has two subroutines: the main

[1]While we believe this description is accurate, unfortunately, due to language issues and some missing technical and experimental details in the cited article, we have been unable to replicate this work.

**Algorithm 2** RECOMPUTE_VALUE

```
Input: Task $i$, Update Interval $k$, Simulation Episodes $Sim$
Output: Recomputed value and completion functions for the task graph below and including $i$
 1: if $Count(i) < k$ then
 2:     return
 3: end if
 4: if $i$ is primitive then
 5:     Sample new transition and reward parameters $\Theta$ from current posterior $\Psi$
 6: else
 7:     for all child tasks $a$ of $i$ do
 8:         RECOMPUTE_VALUE($a$, $k$, $Sim$)
 9:     end for
10:     for $Sim$ episodes do
11:         $s \leftarrow$ random nonterminal state of $i$
12:         Run MAXQ-0($i$, $s$, $\Theta$) {Table 2 in [7]}
13:     end for
14: end if
15: $Count(i) \leftarrow 0$
```

BAYESIAN_MAXQ routine (Algorithm 1) and an auxiliary RECOMPUTE_VALUE routine (Algorithm 2). Intuitively, the main BAYESIAN_MAXQ routine interacts with the world using the current value estimates and updates the posterior over the environment models. Every $k$ steps, it calls the RECOMPUTE_VALUE function. This function then resamples a model from the posterior and recomputes the value and completion functions to reflect this new model. Just as in standard model-based RL, this procedure continues until the hierarchical policy converges. In this description, the value $V$ and completion $C$ functions are assumed to be global.

At the start of each episode, the BAYESIAN_MAXQ routine is called with the $Root$ task and the initial state for the current episode. The standard MAXQ execution protocol is then followed, where each task chooses an action based on its current value function (initially random). When a primitive action is reached and executed, it updates the posterior over model parameters (Line 3) and its own value estimate (which is just the reward function for primitive actions). When a task exits and returns to its parent, the parent subsequently updates its completion function based on the current estimates of the value of the exit state (Lines 13 and 14). Note that in MAXQ, the value function of a composite task can be (recursively) computed using the completion functions of subtasks and the rewards obtained by executing primitive actions, so we do not need to separately store or update the value functions (except for the primitive actions where the value function is the reward). Finally, each primitive action maintains a count of how many times it has been executed and each composite task maintains a count of how many child actions have been taken.

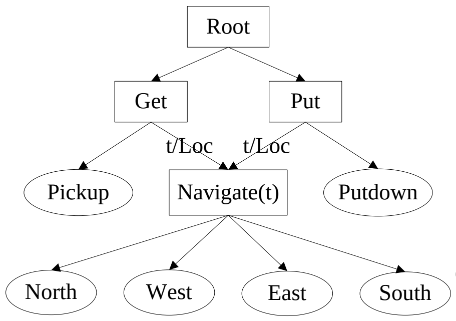
When $k$ (an algorithm parameter) steps have been executed in a composite task, BAYESIAN_MAXQ calls RECOMPUTE_VALUE to re-estimate the value and completion functions (the check on $k$ is shown in RECOMPUTE_VALUE, Line 2). When activated, this function recursively re-estimates the value/completion functions for all subtasks of the current task. At the level of a primitive action, this simply involves resampling the reward and transition parameters from the current posterior over models. For a composite task, we use the MAXQ-0 algorithm (Table 2 in [7]). [2] We run this algorithm for $Sim$ episodes, starting with the current subtask as the root. This algorithm recursively updates the completion function of the task graph below the current task. Note that in this step, the subtasks with primitive actions use model-based updates. That is, when a primitive action is "executed" in such tasks, the currently sampled transition function (part of $\Theta$ in Line 5) is used to find the next state, and then the associated reward is used to update the completion function. This is similar to Lines 12, 13 and 14 in BAYESIAN_MAXQ, except that it uses the sampled model $\Theta$ instead of the real environment. After RECOMPUTE_VALUE terminates, a new set of value/completion functions are available for BAYESIAN_MAXQ to use to select actions.

[2]The MAXQ-0 algorithm assumes there are no *pseudo-rewards*, i.e., preferences over possible exits for a subtask. If this is not true, this can be replaced by the MAXQ-Q algorithm.

We note that in the ideal case, instead of running for $Sim$ episodes, we would simply run MAXQ-0 to convergence; in practice this is not possible, so we impose an arbitrary cutoff. Similarly, ideally we would set $k = 1$. In this case, in fact, Lines 13 and 14 are not needed, since the completion functions would be recomputed every step. Again, this is generally not possible in practice.

## 4 Empirical Evaluation

In this section, we evaluate our approach and test three hypotheses: First, does incorporating model-based priors help speed up the convergence of MAXQ to the optimal policy? Second, since in this case both the prior and the structure of the task hierarchy convey information about the structure of the task, how do they interact? Does the task hierarchy still matter if very good priors are available for primitive actions? Finally, how does Bayesian HRL compare to standard (flat) Bayesian RL? Does Bayesian RL perform better (in terms of computational time) if a task hierarchy is available?



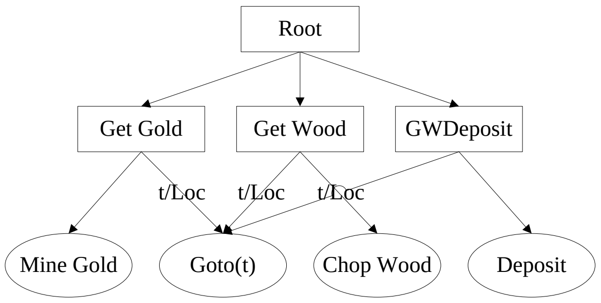


Figure 1: Task Hierarchies for Taxi-world (left) and Resource-collection (right).

To evaluate these hypotheses, we use two domains, two prior distributions and a number of baseline methods. The two domains we use are the well-known Taxi-world [7] and Resource-collection [14]. In Taxi-world, the agent controls a taxi in a grid-world that has to pick up a passenger from a source location and drop them off at their destination. The state variables consist of the location of the taxi and the source and destination of the passenger. The actions available to the agent consist of navigation actions and actions to pickup and putdown the passenger. The agent gets a reward of +20 upon completing the task, a constant -1 reward for every action and a -10 penalty for an erroneous action (e.g. trying to putdown a passenger when it is not carrying one). We use a more challenging variant of this basic task, the "fickle taxi", where each navigation action has a 15% chance of moving in a direction orthogonal to the intended move (so 30% chance total). The Resource-collection domain is inspired by real-time strategy games where one component involves collecting resources from a map in order to build units. We simulate this through a grid world environment where the agent controls a unit that can harvest gold and chop wood. Here the state variables consist of the location of the agent, what the agent is carrying, whether a goldmine or forest is adjacent to its current location and whether a desired gold or wood quota has been met. The actions available to the agent are to move to a specific location, chop gold or harvest wood, and to deposit the item it is carrying (if any). To make the task tractable we limit the set of possible moves to a set of four locations adjacent to each goldmine, forest or deposit location. As before, for each navigation action the agent has a probability of successfully completing it with 0.7 probability and moving to a random location with 0.3 probability. Further, in this domain, there may be multiple goldmines and multiple forests and multiple locations next to each goldmine and forest, so the action space is still quite large. In our experiments, the map contains two goldmines and two forests, each containing two units of gold and two units of wood, and the gold and wood quota is set to three each. The agent gets a +50 reward when it meets the gold/wood quota, a constant -1 reward for every action and an additional -1 if it takes a "bad" action (such as trying to deposit when it is not carrying anything). The task hierarchies corresponding to these domains are shown in Figure 1.

For each domain, for the Bayesian methods, we use Dirichlet priors for the transition function parameters and Normal-Gamma priors for the reward function parameters. We use two different parameter

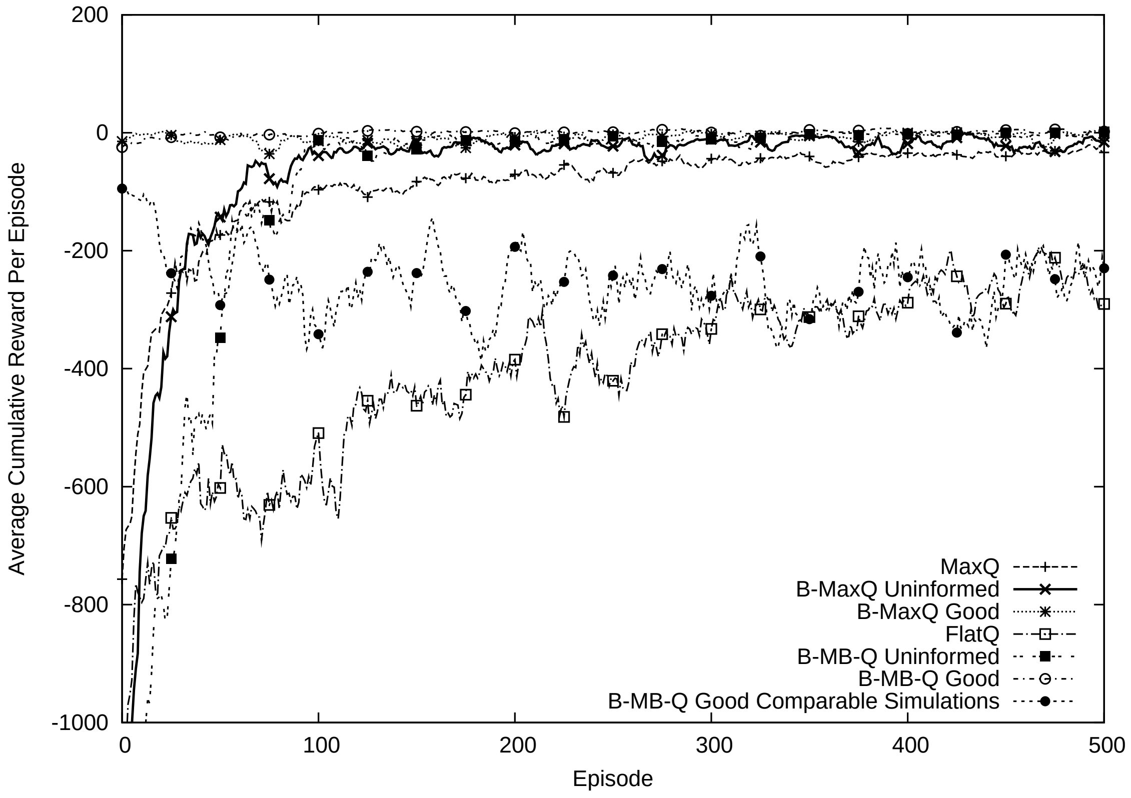


Figure 2: Performance of different algorithms on Taxi-world. The prefix "B-" denotes Bayesian, "Uninformed/Good" denotes the prior and "MB" denotes model-based.

settings for these priors. The first case is an uninformed prior. Here the priors are set to approximate a uniform distribution over model parameters. The second case is a "good" prior. In this case, we first run the Bayesian MAXQ approach for 1000 episodes on each problem, and use the estimated model posterior as the prior. The prior distributions we choose are conjugate to the likelihood, so we can compute the posterior distributions over the relevant parameters in closed form. In general, this is not necessary; more complex priors could be used as long as we can sample from the posterior distribution.

The methods we use in our experiments are: (i) Flat Q, the standard, non-Bayesian Q-learning algorithm, (ii) MAXQ-0, the standard, non-Bayesian Q-learning algorithm for MAXQ with no pseudo-reward, (iii) Bayesian model-based Q-learning with uninformed prior and (iv) with a "good" prior, (v) Bayesian MAXQ (our proposed approach) with a uninformed and (vi) with a "good prior." In our implementation, the Bayesian model-based Q-learning uses the same code as the Bayesian MAXQ algorithm, with a "trivial" hierarchy consisting of the Root task with only the primitive actions as children. We make two changes to the Bayesian MAXQ algorithm for efficiency: first, we use "all states updating" [7], where the completion function is updated fro all states seen in the child task rather than just the exit. Second, in Line 5 in RECOMPUTE_VALUE, we sample a model from the posterior [16]. We observed that, as noted by others [19], the convergence behavior can be improved by sampling an approximately MAP model. We approximate this by computing the MAP model from our posterior, and then using an $\epsilon$-greedy hierarchical policy to select actions. The results of these experiments are shown in Figures 2 and 3. For the Bayesian methods, the update frequency $k$ was set to 50 for Taxi-world and 25 for Resource-collection. $Sim$ was set to 200 for Bayesian MAXQ for Taxi-World and 1000 for Bayesian model-based Q, and to 1000 for both for Resource collection.

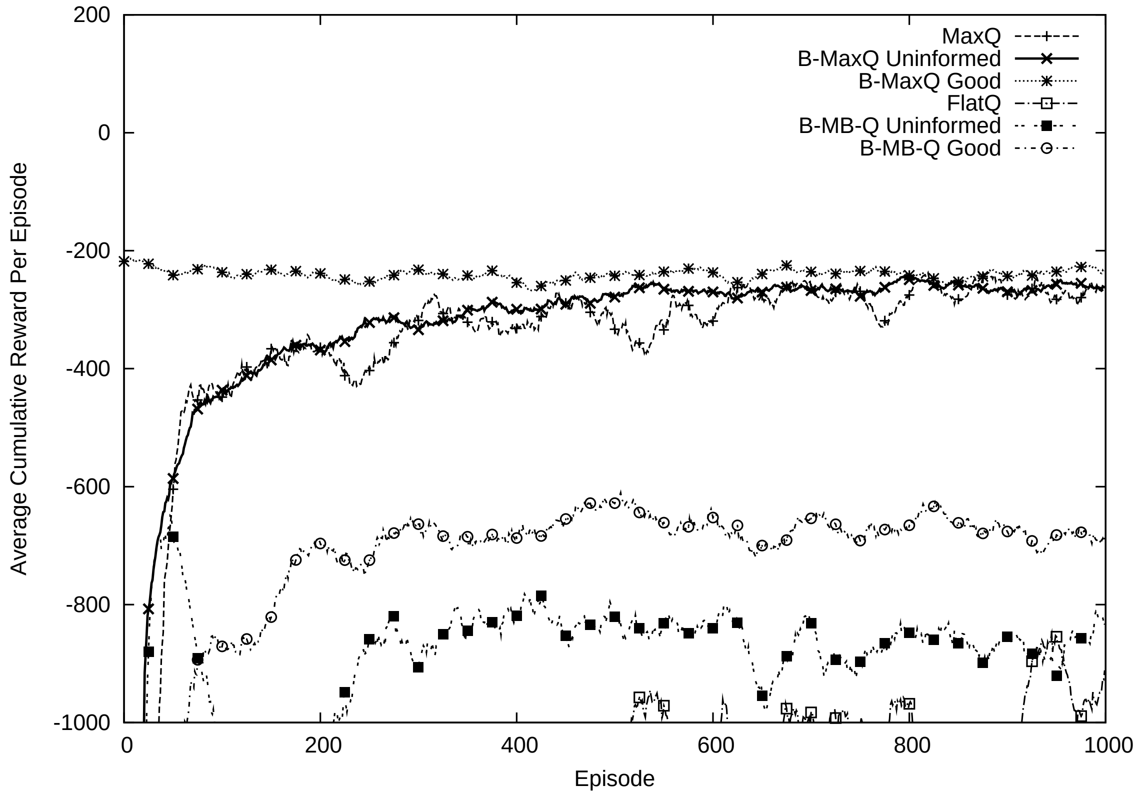


Figure 3: Performance of different algorithms on Resource-collection. The prefix "B-" denotes Bayesian, "Uninformed/Good" denotes the prior and "MB" denotes model-based.

From these results, comparing the Bayesian versions of MAXQ to the non-Bayesian version, we observe that for Taxi-world, the Bayesian version converges faster to the optimal policy even with the uninformed prior, while for Resource-collection, the convergence rates are similar. When a good prior is available, convergence is very fast (almost immediate) in both domains. Thus, the availability of model priors can help speed up convergence in many cases, even for HRL.

Next, we compare the Bayesian MAXQ approach to the "flat" Bayesian model-based Q learning (which is nearly identical except the lack of the task hierarchy). We note that in Taxi-world, with uninformed priors, though the "flat" method initially does worse, it soon catches up to standard MAXQ and then to Bayesian MAXQ. This is probably because in this domain, the primitive models are relatively easy to acquire, and the task hierarchy provides no additional leverage. Note that for the "good" prior in this task, Bayesian MAXQ and flat Bayesian model-based learning do about equally well. This indicates that in some cases the availability of a good model prior may reduce the need for a task hierarchy. However, this does not always happen, as shown in the Resource-collection domain, where even with a good prior, "flat" Bayesian model-based Q does not converge. The difference is that in this case, the task hierarchy encodes extra information that cannot be deduced just from the models. In particular, the task hierarchy tells the agent that good policies consist of gold/wood collection moves followed by deposit moves. Since the reward structure in this domain is very sparse, it is difficult to deduce this even if very good models are available. Thus, good priors may not be sufficient to guarantee fast convergence—task hierarchies can encode additional information about good policies that help convergence even in the presence of good priors. However, if no such policy information is encoded by the hierarchy (e.g. the hierarchy allows all "legal" policies), then a good prior on the model reduces or eliminates the convergence advantage of HRL over flat RL.

Table 1: CPU time taken by various methods on Taxi-world.

| Method | Tot. Time for 500 Episodes (s) | Episodes with 5-20 Steps (near-optimal) | | Episodes with 70-120 Steps (suboptimal) | |
|---|---|---|---|---|---|
| | | Avg Time (ms) | Count | Avg Time (ms) | Count |
| Bayesian MaxQ, Uninformed Prior | 179 | 21 | 81 | 332 | 39 |
| Bayesian Model-based Q, Uninformed Prior | 232 | 162 | 211 | 851 | 25 |
| Bayesian MaxQ, Good Prior | 14 | 17 | 209 | 84 | 3 |
| Bayesian Model-based Q, Good Prior | 119 | 172 | 316 | - | 0 |
| Bayesian Model-based Q, Good Prior & Comparable Simulations | 934 | - | 0 | 687 | 56 |
| MaxQ | 0.53 | - | 0 | 1.49 | 171 |
| Flat Q | 1.15 | - | 0 | 0.37 | 18 |

Finally, we compare the time taken by the different approaches in our experiments in Taxi-world (Table 1). We observe that, as expected, the Bayesian RL approaches are significantly slower than the non-Bayesian approaches. However, out of the Bayesian methods, the Bayesian MAXQ approaches are significantly faster than the flat Bayesian model-based approaches. This can be attributed to the fact that for the flat case, during the simulation in RECOMPUTE_VALUE, a much larger task needs to be solved, while the Bayesian MAXQ approach is able to take into account the structure of the hierarchy to only simulate subtasks as needed, which ends up being much more efficient. However, we note that we allowed the flat Bayesian model-based approach 1000 episodes of simulation as opposed to 200 for Bayesian MAXQ. Clearly this increases the time taken for the flat cases. But at the same time, this is necessary: the “Comparable Simulations” row (and curve in Figure 2) shows that, if the simulations are reduced to 250 episodes for this approach, the resulting values are no longer reliable and the performance of the Bayesian flat approach drops sharply. Thus, taking advantage of the hierarchical task decomposition helps reduce the computational cost of Bayesian RL.

## 5 Conclusion

In this paper, we have proposed an approach to incorporating probabilistic priors on environment models into HRL by extending the MAXQ framework. Our approach uses a combination of model-based and model-free learning to compute a recursively or hierarchically optimal policy. Our experiments indicate that: (i) taking advantage of model priors can speedup convergence for HRL; (ii) however, good priors can reduce or eliminate the convergence advantage of HRL over flat RL, if the hierarchy does not encode “strong” policy constraints, and (iii) the cost of Bayesian RL can be reduced in the HRL setting, because the sampled solutions only involve subtasks rather than the whole task and so can be substantially more efficient. In future work, we propose to investigate methods to incorporate priors on value funcitons into the learning process, as well as investigate multi-task RL scenarios where the priors we currently assume to be given can be learned.

## References

[1] D. ANDRE AND S. RUSSELL, *State Abstraction for Programmable Reinforcement Learning Agents*, in Proceedings of AAAI, 2002.

[2] A. G. BARTO AND S. MAHADEVAN, *Recent advances in hierarchical reinforcement learning*, Discrete Event Dynamic Systems, 13 (2003), pp. 341–379.

[3] R. I. Brafman, M. Tennenholtz, and P. Kaelbling, *R-max - a general polynomial time algorithm for near-optimal reinforcement learning*, Journal of Machine Learning Research, (2001).

[4] Z. Dai, X. Chen, W. Cao, and M. Wu, *Model-based learning with bayesian and maxq value function decomposition for hierarchical task*, in Proceedings of the 8th World Congress on Intelligent Control and Automation, 2010.

[5] R. Dearden, N. Friedman, and D. Andre, *Model based bayesian exploration*, in Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1999.

[6] R. Dearden, N. Friedman, and S. Russell, *Bayesian Q-learning*, in Proceedings of the Fifteenth National Conference on Artificial Intelligence, 1998.

[7] T. G. Dietterich, *Hierarchical reinforcement learning with the maxq value function decomposition*, Journal of Artificial Intelligence Research, 13 (2000), pp. 227–303.

[8] Y. Engel, S. Mannor, and R. Meir, *Bayes meets Bellman:the Gaussian process approach to temporal difference learning.*, in Proceedings of the 20th Internationl Conference on Machine Learning, 2003.

[9] M. Ghavamzadeh and Y. Engel, *Bayesian actor-critic algorithms*, in Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007), Z. Ghahramani, ed., Omnipress, 2007, pp. 297–304.

[10] M. Ghavamzadeh and Y. Engel, *Bayesian policy gradient algorithms*, in Advances in Neural Information Processing Systems 19, MIT Press, 2007.

[11] N. K. Jong and P. Stone, *Hierarchical model-based reinforcement learning: R-max + maxq*, in Proceedings of the 25 th International Conference on Machine Learning, 2008.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research, 4 (1996), pp. 237–285.

[13] A. Lazaric and M. Ghavamzadeh, *Bayesian multi-task reinforcement learning*, in Proc. 27th International Conference on Machine Learning, 2010.

[14] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich, *Automatic discovery and transfer of MAXQ hierarchies*, in Proceedings of the 25th International Conference on Machine Learning, A. McCallum and S. Roweis, eds., Omnipress, 2008, pp. 648–655.

[15] M. Stolle and D. Precup, *Learning Options in reinforcement Learning*, vol. 2371/2002 of Lecture Notes in Computer Science, Springer, 2002, pp. 212–223.

[16] M. J. A. Strens, *A Bayesian framework for reinforcement learning*, in Proceeding of the 17th International Conference on Machine Learning, 2000.

[17] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[18] W. R. Thompson, *On the likelihood that one unknown probability exceeds another in view of the evidence of two samples*, Biometrika, 25 (1933), pp. 285–294.

[19] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, *Multi-task reinforcement learning: a hierarchical bayesian approach*, in ICML '07: Proceedings of the 24th international conference on Machine learning, New York, NY, USA, 2007, ACM, pp. 1015–1022.