Multi-task Value of Information Planning for Sequential Multi-task Bandits

Rika Antonova

Adviser: Emma Brunskill

CMU-RI-TR-16-41

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Keywords: Reinforcement Learning, Bayesian Multi-Armed Bandits, Value of Information

Abstract

In sequential decision making under uncertainty, algorithms and agents that learn across related tasks can have significantly better performance than approaches that neglect to leverage related experience. In this work we consider online learning across a sequence of tasks, where each task is drawn from a finite set of multi-armed bandits (MABs). We introduce the Multi-task Value of Information (MT-VOI) planner, which balances exploration and exploitation at a task level by evaluating the benefits of additional exploration in the current task in order to improve reward across tasks. Our approach demonstrates a substantial improvement over single-task algorithms and a recent multi-task algorithm designed specifically for acting across a sequence of MABs.

Contents

1 Background and Motivation

1.1 Introduction

Algorithms and agents that learn across related tasks can have significantly better performance than approaches that neglect to leverage related experience. In this work we consider online learning across a sequence of tasks, where each task itself involves sequential decision making under uncertainty. For example, consider an adaptive tutoring system with modifiable settings appealing to different types of students. The goal of this system could be both to help the current students learn more effectively and to identify better settings that will help future students. The system could automatically discover optimal settings for various groups of students (e.g. best settings for cautious slow learners, impatient fast learners, students responding best to graphical aids, those preferring verbal explanations, etc). Examples in other domains include: learning to present news or ad recommendations targeted to a group of customers with similar interests (while automatically discovering both the types of groups and the most appropriate articles/ads for each group type), or learning effective control settings for a robot performing tasks in diverse environments (e.g. different control approaches for sandy, rocky, hilly, flat terrains).

An algorithm that selects activities on each time step in order to maximize exactly the expected sum of future rewards across all tasks is very appealing but likely to be computationally intractable. As a step towards this objective, in this work we describe the Multi-task Value of Information (MT-VOI) planner for a sequence of tasks sampled from a finite set of multi-armed bandits (MABs). MT-VOI balances exploration and exploitation at a task level by evaluating if it would be beneficial to perform additional exploration in the current task in order to improve reward across tasks. In empirical simulations we demonstrate a substantial improvement over single-task algorithms and over a recent state-of-the-art approach designed specifically for acting across a sequence of MABs. MT-VOI achieves these improvements despite starting from uninformed prior without any additional domain knowledge. Our experimental results highlight the benefit of explicitly reasoning about future tasks when selecting actions.

1.2 Previous Work

Previously there have been encouraging empirical demonstrations of the benefit of transfer reinforcement learning (see survey by [19]) from a source task to improve performance on a target task, and some recent algorithms that guarantee a performance improvement due to learning across tasks [3, 6, 12]. Prior work by Wilson et al. [20] considered acting in a sequence of tasks sampled from a class of MDPs. The authors maintain a Bayesian posterior over the mixture of MDPs, and update it using approximate inference. When acting in a new task, the presented approach maintains a probability over the current task parameters, but does not seek to do efficient exploration, either within the current task or across tasks, to maximize performance. In contrast to this, Brunskill and Li [6] present a Probably Approximately Correct Reinforcement Learning for a similar setting, but in this work the control approach explicitly leverages the learned finite set of parameters to improve performance in next tasks. Most recently Azar, Lazaric and Brunskill [3], considered online learning across a series of tasks drawn from a finite set of multi-armed bandits. This work

uses a Methods of Moments approach to learn the set of MABs' parameters, and leverages this in an optimism under uncertainty approach to improve performance on the current task.

Despite this previous research, many important questions remain open about how to best learn and leverage information about multi-task structure to enhance performance across a series of tasks. These issues relate to the exploration–exploitation tradeoff so well known in single task decision making under uncertainty, where an agent may risk sacrificing some early rewards in order to learn a better policy with higher later performance. It seems highly likely that a similar exploration-exploitation tradeoff exists *across tasks*.

1.3 Motivation

All the prior approaches in sequential multi-task under uncertainty act to maximize performance on the current task given prior experience. In contrast to previous work, we wish to develop action selection algorithms that maximize performance across all tasks, current and future. First, a natural question is whether thinking about the multi-task nature of the problem is necessary when selecting actions within a particular task. If all tasks are completely unrelated, the multi-task objective function decomposes into maximizing the expected performance in each individual task. However, in many cases there will be some relationship between the tasks. In particular, if all tasks are sampled from the same underlying distribution, there will often be structure we can leverage to improve performance across tasks.

In this work we consider the same setup as that by Azar et al. [3], namely acting in a sequence of J tasks, each of length H, where each task is drawn from a finite set of M multi-armed bandits (MABs), each with N arms. MABs are a simple but powerful framework for sequential decision making under uncertainty. In a MAB, pulling an arm n yields a stochastic payoff sampled from a distribution whose parameters are unknown; we consider the common case of Bernoulli distributions. In our and Azar's multi-task setup, the MAB identity of each new task is unknown, as are the M sets of MAB parameters and the probability distribution (assumed to be a multinomial) of sampling a new task from each of the M MABs. For the examples that were discussed in the introduction, this setup can capture learning the most engaging tutoring system settings for different (latent) types of students, the best type of articles/ads to show to different (latent) types of customers, the appropriate robot control parameters for a (latent) type of terrain.

In this and other related setups, it is reasonable to believe that considering future tasks when acting in the current task may aid overall performance. For example, consider a small illustrative domain with two MABs and three arms as shown in Table 2.

	arm_1	arm_2	arm_3
MAB_1	0.98	0.78	0.90
MAB_2	0.78	0.98	0.02

Table 1: InfoArm domain with a discriminative arm.

In this domain the optimal arm to pull when interacting with MAB_1 is arm_1, when interacting with MAB_2 is arm_2 (probability 0.98 of getting a reward). Despite the fact that arm_3 is not optimal in either of the MABs when the identity of the MAB is known, arm_3 is most informative when trying to distinguish between the two MABs (getting a reward after pulling arm_3 hints strongly

that we are interacting with MAB_1 and not MAB_2). Thus, if there is a large number of tasks to complete, it may be worth better learning the parameters of arm_3 and pulling arm_3 to discriminate effectively between the two different MABs. However, near the end of our task sequence, it may not be worth taking actions to refine the posterior over the arm parameters, because while arm_3 is helpful in distinguishing between the two MABs, it is also costly to explore. In general, the optimal strategy for maximizing the expected sum of future rewards across all tasks will depend on the true underlying MAB parameters, the horizon left in the current task, and the number of tasks remaining.

2 Bayesian Framework for Multi-task MAB Planning

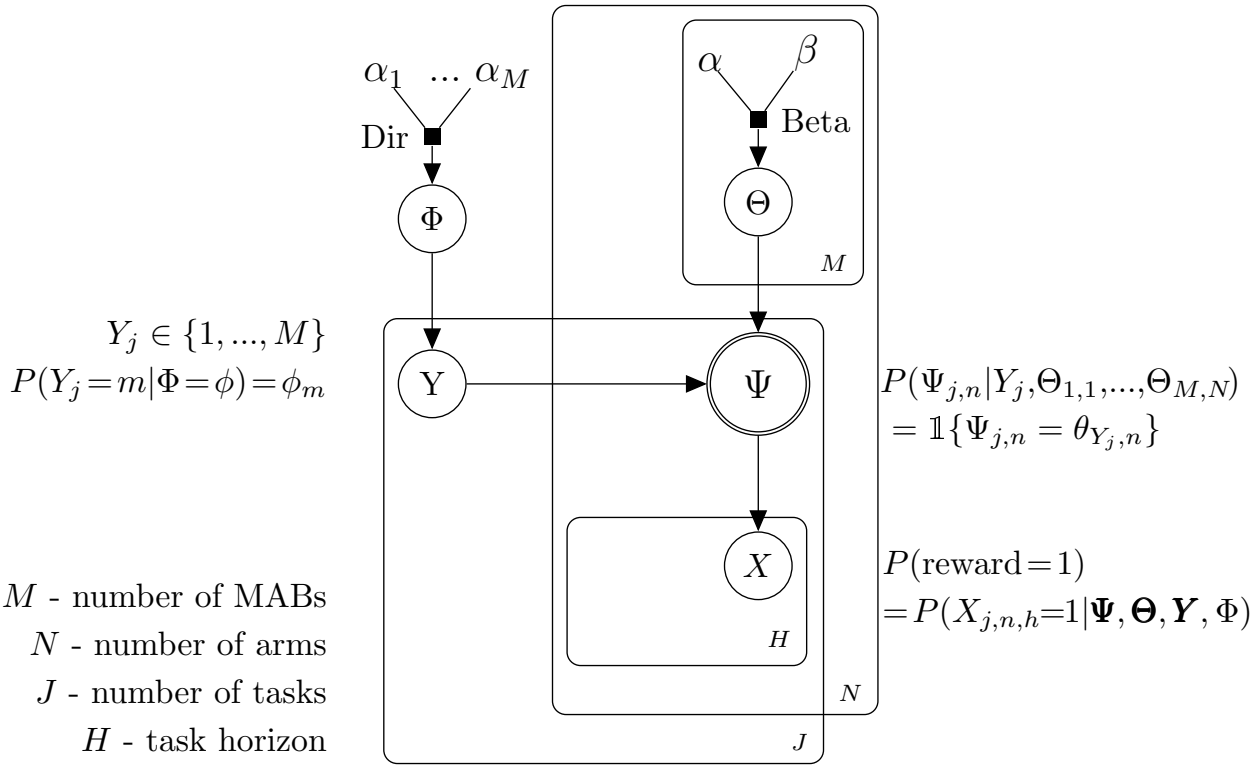
When tasks are sampled from a stationary distribution, the problem of maximizing total expected sum of rewards can be posed as partially observable Markov decision process (POMDP) planning problem. POMDP is a framework for sequential decision making under uncertainty when the state is unobservable – true parameters of the model are never revealed, and are only sensed indirectly through observations. POMDP planning involves computing decisions to maximize the expected sum of future rewards given the history of prior actions and observations (or a sufficient statistic of such). Researchers have noted that it is possible to reframe learning in a single task sequential decision making under uncertainty (where the parameters are unknown) as a POMDP planning problem (e.g. [1, 8, 15]). Drawing inspiration from this work, we will frame our multi-task setting as a special POMDP planning problem. To do so we require both (1) maintaining and updating a Bayesian belief state that summarizes our knowledge of the underlying MAB parameters, the distribution over MABs, and the MAB identity of the current task, and (2) an algorithm for planning in this POMDP to compute which arm to pull given the current belief state. We now describe our approach to (1): using a Bayesian Network representation for our setup to maintain belief state for POMDP planning.

2.1 Bayesian Network for Belief State

Recall from section 1.3 that in this work we consider acting in a sequence of J tasks, each of length H, where each task is drawn from a finite set of M multi-armed bandits (MABs), each with N arms. We pose our problem as planning in a special POMDP, where the latent state is the joint probability distribution over MABs, the payoffs of each MAB, and the MAB identity of the current task. The transition function over hidden states is simple: the underlying task parameters are static (while interacting with the same task), and after H steps the identity of the current task is resampled. The action space is the original action space (the set of arms) and the observations are the rewards. Selecting an action then involves performing POMDP planning across a total horizon of $J \cdot H$ steps (J tasks, each of H steps).

Upon a closer look at our setup it becomes clear that the dependencies between latent states and rewards could be captured by a Bayesian Network. Bayesian Network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies with a directed acyclic graph (see [10] for an overview of probabilistic graphical models and their properties). Bayesian Network for our setup is shown in Figure 1. The network is in plate notation (see section 6.4.1 of [10]), where instead of drawing each variable individually, a rectangular plate is used

Figure 1: Bayesian network for multi-task belief state.



to represent variables with identical distributions, and the number at the bottom of the plate represents the number of necessary repetitions (M MABs, N arms each, J tasks, H steps per task). We will now describe each of the random variables involved in Bayesian Network of Figure 1, from top to bottom.

Φ : Dirichlet-distributed random variable that expresses our belief about relative frequencies of each MAB identity among the tasks (in our work "MAB identity" and MAB with a particular set of parameters are taken to be synonymous, since each MAB identity represents a type/cluster/kind of agent and is expressed by a MAB with certain concrete reward probabilities for each arm). $\alpha_1, ..., \alpha_M$ serve as initial parameters for Φ and capture our prior belief of the prevalence of each MAB.

Y_j : latent random variable that represents the identity of the MAB we interact with in j^{th} task. Y_j is latent, because MAB identity (the set of true MAB parameters) in our setup is never observed – after the task is over we are not informed about the true parameters of the MAB we have just interacted with. The probability distribution of sampling a new task from each of the M MABs is assumed to be a multinomial in our setup, and so Y_j is a multinomial random variable, with Φ serving as its Dirichlet prior.

$\Theta_{m,n}$: Beta-distributed random variable capturing distribution of reward parameter for the n^{th} arm in MAB m. In our setup arm payoff distribution is Bernoulli, so $\Theta_{m,n}$ serves as Beta prior

for the payoff of arm n of MAB m. α, β are used as initial parameters if prior information is available about MAB payoffs.

$\Psi_{j,n}$: multiplexer variable to "select" $\theta_{Y_j,n}$ that will generate rewards for the current arm pull (see Definition 5.3 in [10] for details on multiplexers). The value of $\Psi_{j,n}$ is effectively a copy of the value of one of its parents, $\Theta_{1,1}, ..., \Theta_{M,N}$. Its probability mass function (PMF) can be thought of as an indicator function that returns 1 when input corresponding to the parameters of the MAB of the current task is given, and returns 0 otherwise.

$X_{j,n,h}$: Bernoulli random variable that expresses the probability of getting a reward when acting in j^{th} task, pulling arm n on step h. $X_{j,n,h}$ are the only observed variables in this Bayesian Network (we know the value of the reward after we pull an arm, but we never learn the true parameters of the MABs or MAB identities involved in the tasks).

Plate notation for Bayesian Network offers a concise graphical representation of the network, and in the same spirit we define vector notation for the set of random variables located in the same plate. Let $\boldsymbol{\Theta}$ be a vector composed of $M{\cdot}N$ elements $\Theta_{1,1}, .., \Theta_{M,N}$, and let $\boldsymbol{\theta}$ be a vector composed of the values $\theta_{1,1}, .., \theta_{M,N}$ to be assigned to the corresponding random variables. Similarly, let $\boldsymbol{\Psi}$ be a vector composed of $J \cdot N$ elements $\Psi_{1,1}, .., \Psi_{J,N}$, let $\boldsymbol{\psi}$ be a vector of corresponding values $\psi_{1,1}, .., \psi_{J,N}$; also let $\boldsymbol{Y}$ be a vector composed of J elements $Y_1, ..., Y_J$, and let $\boldsymbol{y}$ be a vector of corresponding values $y_1, ..., y_J$.

Let us also outline notation we use for denoting random variables, their probability density/mass functions and expectations:

- capital greek letters and X, Y are used to denote random variables
- lower case greek letters and x, y denote the concrete values of random variables
- $f_X(x)$ denotes probability density function (PDF) of X if X is a continuous random variable; $P(X{=}x)$ and $P(x)$ denote probability mass function (PFM) if X is a discrete
- $f_{X,Y}(x, y)$ denotes PDF of a joint distribution of X and Y
- to obtain a marginal distribution of X (marginalizing out variable Y) we write: $\int_y f_{X,Y}(x, y)dy$ $\Big($or for discrete variables: $\sum_y f_{X,Y}(x, y) = \sum_y P(x, y)\Big)$
- $E_Y[X]$ denotes expectation over Y of a random variable X: $\int_y x f_{X,Y}(x, y)$
- using vector notation introduced in the previous paragraph: $\int_{\boldsymbol{\theta}} f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})d\boldsymbol{\theta} = \int_{\theta_{1,1}} \cdots \int_{\theta_{M,N}} f_{\Theta_{1,1}}(\theta_{1,1}) \cdots f_{\Theta_{M,N}}(\theta_{M,N})d\theta_{1,1} \cdots d\theta_{M,N}$
- to integrate over $\boldsymbol{\Theta}$ excluding integrating over $\Theta_{m,n}$ we write: $\int_{\boldsymbol{\theta}_{\setminus\theta_{m,n}}} f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})d\boldsymbol{\theta}_{\setminus\theta_{m,n}}$ (the result is a function of $\Theta_{m,n}$, since it is the only variable that was not marginalized out)

2.2 Bayesian Multi-task Belief Updating

Below we describe the process for updating belief state (posterior distribution) while acting within a single task, as well after a task completes. Our choice of using conjugate priors enables all calculations to be performed analytically, which is a nice advantage in contrast to some related work by Wilson et al. [20] on (a more general formulation of) acting in a finite set of MDPs, where the authors used approximate inference to update the multi-task posterior.

The general procedure for deriving Bayesian updates is the same in spirit for each of the conditional probability distributions (random variables) in our Bayesian Network:

1. start with the joint distribution of all the random variables in the Bayesian network
2. using the joint distribution write down the expression for the probability of getting a reward (or a sequence of rewards)
3. carefully marginalize, leaving only the desired variable – this process requires taking advantage of the conditional independencies encoded by the Bayesian network

2.3 Belief Updating Within the Same Task

Let us first demonstrate how to compute the probability of obtaining a reward after a sequence of observations in the same task. Suppose we started the first task (sampled a MAB $m, m \in \{1...M\}$) and observed r_n rewards of 1 and z_n rewards of 0 for each of $n = 1...N$ arms. For example if we pulled arm 1 and received no reward (got a "reward" of 0) and then pulled arm 2 twice and received a reward of 1 both times, we will have $r_1 = 0, r_2 = 2, z_1 = 1, z_2 = 0$. Because pulling arms does not change the underlying MAB parameters, the number of rewards of 1 and 0 obtained from each arm is a sufficient statistic when recording our interaction with a MAB (within the same task). Hence we summarize the assignment of values to $X_{j,n,h}$ variables as $\boldsymbol{X} = \{\hat{r}, \hat{z}\}$, where $\hat{r} = \{r_1, ..., r_N\}, \hat{z} = \{z_1, ..., z_n\}$.

Now let $\boldsymbol{X} = \{\hat{r}_{(r_n++)}, \hat{z}\}$ denote the event that we obtain a reward of 1 from arm n on the next pull. Then the joint probability of observing a collection of rewards summarized by $\hat{r}_{(r_n++)} = \{r_1, ..., r_n+1, ..., r_N\}, \hat{z} = \{z_1, ..., z_n\}$ is:

$$P(\boldsymbol{X}=\{\hat{r}_{(r_n++)}, \hat{z}\}, \boldsymbol{\Psi}, Y_1, \Phi, \boldsymbol{\Theta}) = P(\boldsymbol{X}=\{\hat{r}_{(r_n++)}, \hat{z}\}|\boldsymbol{\Psi})\ P(\boldsymbol{\psi}|Y_1, \Phi, \boldsymbol{\Theta})\ P(y|\Phi)\ f_\Phi(\phi)\ f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})$$

Note that we are able break the joint distribution into a product of conditional distributions by using independencies encoded by the Bayesian Network in Figure 1 (for more details see Appendix A). We can now use variable elimination (see section 9.3 of [10]) to compute $P(\boldsymbol{X} = \{\hat{r}_{(r_n++)}, \hat{z}\})$. To make it easier to follow transformations we underline parts of equations that are about to be significantly rearranged and highlight the results.

$$P(\boldsymbol{X}=\{\hat{r}_{(r_n++)}, \hat{z}\}) = \sum_{y=1}^{M} \sum_{\psi} \int_{\phi} \int_{\boldsymbol{\theta}} P(\boldsymbol{X}=\{\hat{r}_{(r_n++)}, \hat{z}\}, \boldsymbol{\Psi}, Y_1, \Phi, \boldsymbol{\Theta}) d\phi\ d\boldsymbol{\theta} \tag{1}$$

$$= \sum_{y=1}^{M} \int_{\phi} \int_{\boldsymbol{\theta}} \underline{P(r_1, z_1|\Theta_{y,1}) \cdots P(r_n+1, z_n|\Theta_{y,n}) \cdots P(r_N, z_N|\Theta_{y,N})} \underline{P(y|\Phi) f_\Phi(\phi)} \underline{f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})}\ d\phi\ \underline{d\boldsymbol{\theta}}$$

$$= \sum_{y=1}^{M} \int_{\phi} P(Y{=}y|\Phi) f_{\Phi}(\phi) d\phi \int_{\boldsymbol{\theta}} P(r_1, z_1|\Theta_{y,1}) \cdots P(r_n{+}1, z_n|\Theta_{y,n}) \cdots P(r_N, z_N|\Theta_{y,N}) f_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

$$= \sum_{y=1}^{M} \int_{\phi} \phi_y f_{\Phi}(\phi) d\phi \int_{\theta_{y,1}} P(r_1, z_1|\Theta_{y,1}) f_{\Theta_{1,1}}(\theta_{1,1}) d\theta_{y,1} \cdots \int_{\theta_{y,n}} P(r_n{+}1, z_n|\Theta_{y,n}) f_{\Theta_{y,n}}(\theta_{y,n}) d\theta_{y,n} \cdots$$

$$= \sum_{y=1}^{M} E_{\Phi}[\Phi_y] \cdot E_{\Theta y,n}[\Theta_{y,n}^{r_n+1}(1-\Theta_{y,n})^{z_n}] \cdot \prod_{\substack{i=1 \\ i \neq n}}^{N} E_{\Theta y,i}[\Theta_{y,i}^{r_i}(1-\Theta_{y,i})^{z_i}]$$

The transformation from the first to the second line of 1 a simple re-arrangement. From the second to third line we used the fact that $P(Y_j = m|\Phi = \phi) = \phi_m$: the probability of selecting MAB identity m for the j^{th} task is given by the m^{th} component of dirichlet-distributed variable Φ (which has M components). To go from the third to the fourth line we recognize several expressions that correspond to expectations. $E_{\Phi}[\Phi_y]$ is obtained using the fact that $E[g(\Phi)] = \int_{\phi} g(\phi) f_{\Phi}(\phi) d\phi$ and viewing taking y^{th} component of Φ as a function $g(\Phi)$. Let us adopt concise notation:

$$w_{\Phi_y} = E_{\Phi}[\Phi_y] \tag{2}$$

Expressions of the form $E_{\Theta y,i}[\Theta_{y,i}^{r_i}(1-\Theta_{y,i})^{z_i}]$ can be viewed as special higher moments of $\Theta_{y,i}$, we show derivations for evaluating them in Appendix B. From now on let us adopt a concise notation for these special higher moments:

$$\mu_{\Theta_{y,i}}^{r_i,z_i} = E_{\Theta y,i}[\Theta_{y,i}^{r_i}(1-\Theta_{y,i})^{z_i}] \tag{3}$$

$$w_{\mu_{\Theta_y \backslash n}} = \prod_{i=1, i \neq n}^{N} \mu_{\Theta_{y,i}}^{r_i,z_i} \tag{4}$$

Our original question on how to compute the probability of obtaining a reward of 1 from arm n after a sequence of observations can now be answered by conditioning on observed rewards:

$$P(\text{reward of 1 if arm } n \text{ is pulled next}|\boldsymbol{X} = \{\hat{r}, \hat{z}\}) \tag{5}$$
$$= \frac{P(\boldsymbol{X} = \{\hat{r}_{(r_n{+}{+})}, \hat{z}\})}{P(\boldsymbol{X} = \{\hat{r}, \hat{z}\})} = \frac{\sum_{y=1}^{M} w_{\Phi_y} \cdot w_{\mu_{\Theta_y \backslash n}} \cdot \mu_{\Theta_{y,n}}^{r_n+1,z_n}}{\sum_{y=1}^{M} w_{\Phi_y} w_{\mu_{\Theta_y}}}$$

It is also useful to compute probability of getting a reward of 1 on the next pull of arm n (while still in the same task j) given previous rewards $\hat{r}, \hat{s}$ and given MAB identity m (this will be used for planning algorithms described later for computing lower/upper bounds of payoffs for arms in each MAB):

$$P(\text{reward of 1 if arm } n \text{ is pulled next}|\hat{r}, \hat{z}, Y_j{=}m) = \frac{\cancel{w_{\Phi_y}} \cdot w_{\mu_{\Theta_y \backslash n}} \cdot \mu_{\Theta_{m,n}}^{r_n+1,z_n}}{\cancel{w_{\Phi_y}} w_{\mu_{\Theta_y}}} \tag{6}$$

2.4 Belief Updating upon Task Transition

We are now ready to derive Bayesian updates for Φ and $\boldsymbol{\Theta}$ when transitioning to a new task (let us suppose we have just finished task j). Beta-distributed variables $\Theta_{m,n}$ (with $m \in \{1...M\}, n \in \{1...N\}$) will capture what we have learned from task j about rewards from MAB m, arm n; these Beta-distributed variables will become Beta mixtures after task completion. Intuitively, the reason

for obtaining the mixture is that each Beta component in the mixture corresponds to the version of posterior Beta one would obtain with a certain assignment of arm pulls to MABs. Since we do not observe the identity of the MABs, we only have a probabilistic estimate of the likelihood of each assignment. Hence the posterior takes the form of weighted combination of Beta components, and the weights reflect posterior likelihood of the arm pulls being assigned correctly. The derivation for the posterior mixture is provided below:

$$f_{\Theta_{m,n},\boldsymbol{X}}(\Theta_{m,n}{=}\theta, \boldsymbol{X}{=}\{\hat{r},\hat{z}\}){=}\sum_{y=1}^{M}\sum_{\psi}\int_{\phi}\int_{\boldsymbol{\theta}_{\setminus\theta_{m,n}}} P(\hat{r},\hat{z}|\Psi)P(\Psi|Y_j,\boldsymbol{\Theta})P(y|\Phi)f_{\Phi}(\phi)f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})d\phi d\boldsymbol{\theta}_{\setminus\theta_{m,n}}$$

$$= \sum_{y=1}^{M}\int_{\phi}\int_{\boldsymbol{\theta}_{\setminus\theta m,n}} P(r_1,z_1|\Theta_{y,1})\cdots P(r_N,z_N|\Theta_{y,N})P(y|\Phi)f_{\Phi}(\phi)f_{\Theta_{1,1}}(\theta_{1,1})\cdots f_{\Theta_{M,N}}(\theta_{M,N})d\phi\, d\boldsymbol{\theta}_{\setminus\theta m,n}$$

$$= \sum_{y=1}^{M}\int_{\phi} P(y|\Phi)f_{\Phi}(\phi)d\phi \int_{\boldsymbol{\theta}_{\setminus\theta m,n}} P(r_1,z_1|\Theta_{y,1})\cdots P(r_N,z_N|\Theta_{y,N})f_{\Theta_{1,1}}(\theta_{1,1})\cdots f_{\Theta_{M,N}}(\theta_{M,N})d\boldsymbol{\theta}_{\setminus\theta m,n}$$

$$= \left(E_{\Phi}[\Phi_m]\prod_{\substack{i=1,\\ i\neq n}}^{N}\mu_{\Theta_{m,i}}^{r_i,z_i}\right)P(r_n,z_n|\Theta_{m,n})f_{\Theta_{m,n}}(\theta) + \left(\sum_{\substack{y=1,\\ y\neq m}}^{M}E_{\Phi}[\Phi_y]\prod_{i=1}^{N}\mu_{\Theta y,i}^{r_i,z_i}\right)f_{\Theta_{m,n}}(\theta),$$

using notation from equations 2 and 3:

$$= w_{\Phi_y}w_{\mu_{\Theta_y\setminus n}}\cdot P(r_n,z_n|\Theta_{m,n})f_{\Theta_{m,n}}(\theta) + \sum_{y=1,y\neq m}^{M} w_{\Phi_y}w_{\mu_{\Theta_y}}\cdot f_{\Theta_{m,n}}(\theta) \tag{7}$$

If $\Theta_{m,n}\sim Beta(\alpha,\beta)$, then from $P(r_n,z_n|\Theta_{m,n})f_{\Theta_{m,n}}(\theta)$ we obtain:

$$P(r_n,z_n|\Theta_{m,n}=\theta)f_{\Theta_{m,n}}(\Theta_{m,n}=\theta) = \theta^{r_n}(1-\theta)^{z_n}\cdot\frac{\Gamma(\alpha{+}\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha\text{-}1}(1-\theta)^{\beta\text{-}1}$$

$$= \underbrace{\left(\frac{\Gamma(\alpha{+}\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(r_n{+}\alpha)\Gamma(z_n{+}\beta)}{\Gamma(r_n{+}\alpha{+}z_n{+}\beta)}\right)}_{\text{scalar weight } w \text{ computed using } \Gamma \text{ function}}\underbrace{\frac{\Gamma(r_n{+}\alpha{+}z_n{+}\beta)}{\Gamma(r_n{+}\alpha)\Gamma(z_n{+}\beta)}\theta^{r_n+\alpha\text{-}1}(1-\theta)^{z_n+\beta\text{-}1}}_{\text{PDF of a random variable } \mathcal{B}^{+}\sim Beta(\alpha+r_n,\beta+z_n)} \tag{8}$$

Hence $P(r_n,z_n|\Theta_{m,n})f_{\Theta_{m,n}}(\theta_{m,n})$ is a product of weight w and PDF of a Beta-distributed random variable with parameters $\alpha{+}r_n, \beta{+}z_n$. So we see that after update $\Theta_{m,n}$ becomes a mixture of Beta-distributed components (the summands in equation 7). If $\Theta_{m,n}$ is already a mixture of K components before the update, then we can repeat the steps outlined in equation 8 for each of the K components to obtain w_k and $\mathcal{B}_k^{+}$ for each $k = 1...K$ (where $\mathcal{B}_k^{+}\sim Beta(\alpha_k{+}r_n, \beta_k{+}z_n)$ and α_k, β_k are the parameters of the Beta-distributed variable represented by the k^{th} component of $\Theta_{m,n}$).

Putting together the results of equations 7 and 8, our Bayesian update for $\Theta_{m,n}$ becomes:

$$f_{\Theta_{m,n}}(\theta) \leftarrow f_{\Theta_{m,n}}(\theta|\hat{r},\hat{z}) = \frac{1}{P(\hat{r},\hat{z})}\left[w_{\Phi_m}w_{\mu_{\Theta_m\setminus n}}\sum_{k=1}^{K}w_k f_{\mathcal{B}_k^{+}}(\theta) + \left(\sum_{\substack{y=1,\\ y\neq m}}^{M}w_{\Phi_y}w_{\mu_{\Theta_y}}\right)f_{\Theta_{m,n}}(\theta)\right] \tag{9}$$

Note that products $\prod \mu_{\Theta_{y,i}}^{r_i,z_i}$ capture information from all arms, since we can not assume independence of different arms when doing Bayesian updates across tasks, i.e. $f_{\Theta_{m,n}}(\theta|\hat{r},\hat{z}) \neq f_{\Theta_{m,n}}(\theta|r_n,s_n)$.

Now let's follow the derivation for Bayesian update rule for Φ:

$$f_{\Phi,\boldsymbol{X}}(\Phi=\phi,\boldsymbol{X}=\{\hat{r},\hat{z}\}) = \sum_{y=1}^{M}\sum_{\psi}\int_{\boldsymbol{\theta}} P(\hat{r},\hat{z}|\Psi)P(\Psi|Y_j,\boldsymbol{\Theta})P(y|\Phi)f_{\Phi}(\phi)f_{\boldsymbol{\Theta}}(\boldsymbol{\theta})d\phi\, d\boldsymbol{\theta}$$
$$= \sum_{y=1}^{M}\Big(\prod_{i=1}^{N}\mu_{\Theta_{y,i}}^{r_i,z_i}\Big)\phi_y f_{\Phi}(\phi) = \sum_{y=1}^{M} w_{\mu_{\Theta_y}}\phi_y f_{\Phi}(\phi) \tag{10}$$

We observe that after the completion of the first task Φ becomes a mixture of Dirichlets. Intuitively, each Dirichlet component in the mixture corresponds to a particular assignment of MAB identities in each task; the weight of each mixture reflects the posterior likelihood of the particular assignment:

$$f_{\Phi}(\phi;\alpha_1,...,\alpha_M) = \frac{\Gamma(\sum_{i=1}^{M}\alpha_i)}{\prod_{i=1}^{M}\Gamma(\alpha_i)}\prod_{i=1}^{M}\phi_i^{\alpha_i-1}$$

$$\phi_y f_{\Phi}(\phi) = \underbrace{\left(\frac{\Gamma(\sum_{i=1}^{M}\alpha_i)}{\prod_{i=1}^{M}\Gamma(\alpha_i)}\frac{\Gamma(1+\sum_{i=1}^{M}\alpha_i)}{(\alpha_y{+}1)\prod_{\substack{i=1\\ i\neq y}}^{M}\Gamma(\alpha_i)}\right)}_{\text{scalar weight } w_y \text{ computed using } \Gamma \text{ function}} \underbrace{\frac{(\alpha_y{+}1)\prod_{\substack{i=1\\ i\neq y}}^{M}\Gamma(\alpha_i)}{\Gamma(1+\sum_{i=1}^{M}\alpha_i)}\phi_y^{\alpha_y}\prod_{\substack{i=1\\ i\neq y}}^{M}\phi_i^{\alpha_i-1}}_{\text{PDF of random variable } \Phi_y^{+}\sim Dir(\alpha_1,...,\alpha_y{+}1,...,\alpha_M)} \tag{11}$$

Later, during j^{th} task, Φ could be a mixture of K Dirichlets: $\phi_y f_{\Phi}(\phi) = \phi_y\sum_{k=1}^{K} f_{\Phi_k}(\phi)$, where each component $\Phi_k \sim Dir(\alpha_{k_1},...,\alpha_{k_M})$. In this case using equation 11 we can obtain the corresponding w_{y_k} and $\Phi_{y_k}^{+}$ for each of the K components of the mixture. Hence Bayesian updates would result in adding new components to the mixture but Φ will remain a mixture of Dirichlet-distributed random variables. Notice that each new Dirichlet component contributes a Dirichlet with one of the M parameters incremented. This corresponds to a hypothesis that, in the task we have just completed, we interacted with MAB whose corresponding parameter is incremented (akin to incrementing a counter of how many times we could have encountered this MAB).
Putting together the results of equations 10 and 11, our Bayesian update for Φ becomes:

$$f_{\Phi}(\phi) \leftarrow f_{\Phi}(\phi|\hat{r},\hat{z}) = \frac{1}{P(\hat{r},\hat{z})}\sum_{y=1}^{M} w_{\mu_{\Theta_y}}\sum_{k=1}^{K} w_{y_k} f_{\Phi_{y_k}^{+}}(\phi) \tag{12}$$

2.5 Managing Belief State Representation Growth

The challenge of using the proposed analytical updates in practice comes from the fact that the sizes of the Dirichlet and Beta mixtures grow with the number of tasks completed. This is because the true MAB identity of the tasks is never directly provided, hence the need to retain probability estimates for all the possible assignments of tasks to MABs. The size of the mixture of Betas representing PDF for $\Theta_{m,n}$ grows exponentially with the number of tasks completed: after j tasks the mixture has M^j Beta components (M is the number of different MAB identities and is a fixed

constant). The size of the mixture of Dirichlets that captures PDF for Φ grows polynomially with the number of tasks completed as $O(j^M)$. The smaller order of growth is due to that fact that some Dirichlet components can be combined, reducing the number of components in the mixture; the derivation for this is in Appendix C.1. This kind of belief representation growth is not unique to our setup and has been observed and tackled in previous work ([16], [5], [14]). As we accumulate multi-task evidence and our mixtures grow, we can chose to retain only the components with non-negligible weights. Such approach is most similar in spirit to the "Most Probable" approximation used by [16] to keep belief representation tractable during partially observable MDP RL by only retaining K most probable states. This approach was shown to minimize L_1 distance between exact belief b and approximate belief b', which directly yields a bound $|V^*(b) - V^*(b')|$ between the values of the exact and approximate belief for infinite-horizon tasks and achieves good empirical performance (sec. 5.1 of [16]).

It is also possible to see intuitively why in our setting the approach of discarding mixture components with negligible weights is effective for reducing belief state representation without sacrificing significantly the accuracy of the multi-task posterior. The reasoning is that for MABs with sufficiently different parameters and tasks with horizon greater than $\approx$10 steps, at the end of the task it is likely to be highly certain about the identity of the MAB of the current task. Thus after multi-task posterior is updated upon finishing a task, only one set of Beta components would receive significant weights (those corresponding to the hypothesis of actions and rewards coming from the most likely a-posteriori MAB), while the rest would be updated with small weight factors. As evidence accumulates, mixtures $\Theta_{m,n}$ will more and more closely reflect true MAB payoff distributions. The weights associated with Beta components that are most consistent with the true reward distribution will grow, while the weights associated with other Betas will decrease. Similar reasoning applies to the Dirichlet components of Φ.

3 Multi-task Planning

3.1 Challenges in Multi-task Planning

As discussed earlier, we can frame acting across a series of tasks sampled from a finite set of MABs as a POMDP planning problem, yielding in principle a policy that maximizes the expected sum of rewards across all tasks. However, the hidden states of such POMDP are reward probabilities of the underlying MABs, and are therefore continuous, making this a continuous-state POMDP. Discretizing parameter values can lead to substantial approximations unless it is done to a fine precision, but as the state space grows exponentially with the number of parameters, such representations often quickly exceed the reach of high performing offline discrete-state POMDP approaches such as SARSOP [11].

An alternative approach is to build on recent online tree based methods which have successfully scaled to very large single-task problems. Such approaches only compute an action for the current step, and then perform additional planning after the action is taken and an observation and reward are received. One example is Real-Time Belief Space Search (RTBSS) introduced by [13, 18]. RTBSS progressively builds a tree by exploring reachable belief states starting from the current belief state. The tree growth is limited by pruning the branches that do not lie within the bounds specified for the desired return/reward. Hence the algorithm is able to search deeper,

while avoiding expanding all nodes in the search tree. Another option for online planning is Monte Carlo TreeSearch (MCTS) [7]. For this approach a tree policy is defined to select the most promising node to expand next. Then a simulation is run from the selected node using a simulation policy defined for taking actions in each subsequent state. The search tree is updated using the results/rewards collected during the simulation. An advantage of MCTS is that the values of the intermediate states do not have to be estimated. The result/reward for the simulation can be estimated (if necessary) from only the final state reached. This aspect of MCTS is important for success with domains involving long-horizon tasks with delayed rewards. However, in our domain of multi-armed bandits the difficulty of estimating rewards at each step is similar for all steps, and the reward is received immediately (after pulling an arm). Hence MCTS is unlikely to provide a clear advantage over RTBSS in the setting we consider.

In theory, we could extend a standard forward-search tree POMDP planner to include all of the task transitions in the search tree. However, in order to perform exact planning across all tasks, one would need to build a tree of depth $H \cdot J$ (where H is the number of actions in each task and J is the number of tasks). For domains with N possible actions and Z possible observations (in our case of Bernoulli bandits Z=2), exact planning would involve a tree with $(N \cdot Z)^{H \cdot J}$ nodes, which would be intractable for even the smallest domains. For instance, exact planning for a domain with two MABs, 5 arms, a task horizon of 10, and 8 tasks total would require building a tree with 10^{80} nodes.

One of the key benefits we would hope to get from exact multi-task planning would be enabling more exploratory behavior in earlier tasks, if such behavior would lead to better overall rewards over the full horizon of J tasks by quickly refining our multi-task posterior. This challenge of task-level exploration-exploitation is very similar to the within task exploration-exploitation tradeoff well known in Reinforcement Learning. One approach to handling this in the single task setting, in cases when it is computationally prohibitive to perform even single task exact planning, is to consider the Value of Information (VOI) [17]. Intuitively, VOI represents the benefit of taking an exploratory action in terms of the immediate reward gained by making a more informed decision on the subsequent time step, as compared to the reward of simply taking actions expected to maximize the reward given the current information. Inspired by this, we extend this idea to the task-level VOI. We consider the benefit of executing a more exploratory policy in an entire task, in order to refine the multitask posterior and obtain better reward on the subsequent task, versus myopically maximizing the single task performance of two consecutive tasks. Note that this procedure is a lower bound on VOI on a task level, because we do not consider the benefit of the gained information on tasks farther away then the next two tasks, which makes it a conservative VOI estimate.

3.2 Multi-task Value of Information Planning

We are now ready to introduce our complete multi-task VOI (MT-VOI) for acting across a series of tasks drawn from the same stationary distribution. Again, we emphasize that MT-VOI could also be used for MDPs or other sequential decision making multi-task setups. The pseudocode is outlined in Algorithm 1.

Once a new task is obtained from the environment, we calculate the task-level VOI of acting in current task using an exploratory policy, versus acting only to maximize expected rewards within a

Algorithm 1 : MT-VOI

Set n_{smpl} to a constant of choice
Initialize multi-task posterior b
for $j \leftarrow 1$ **to** J **do**
    Task j is obtained from the environment
    Using the current multi-task posterior b:
    $VOI_j \leftarrow$ **VOIEstimate**(b, n_{smpl})
    if VOI_j *is positive*
    then Act in task j using exploratory policy ;
    else Act in task j to maximize expected reward within a single task (e.g. with RTBSS) ;
    Update the multi-task posterior b
end

Algorithm 2 : **VOIEstimate** (b, n_{smpl})

$VOIestimate \leftarrow 0$
for $i \leftarrow 1$ **to** n_{smpl} **do**
    // Sample from current posterior
    $(\phi_i, \theta_i) \leftarrow$ **SampleParams**(b)
    // Explore, then exploit
    $(r_{expl0}, b'_{expl}) \leftarrow$ **ExploreSingleTask**(ϕ_i, θ_i, b)
    $r_{expl1} \leftarrow$ **RTBSS**$(\phi_i, \theta_i, b'_{expl})$
    // Compare to no exploration
    $(r_0, b') \leftarrow$ **RTBSS**(ϕ_i, θ_i, b)
    $r_1 \leftarrow$ **RTBSS**(ϕ_i, θ_i, b')
    // Compute VOI of exploration
    $VOI_i \leftarrow (r_{expl0}+r_{expl1}) - (r_0+r_1)$
    $VOIestimate \leftarrow VOIestimate + VOI_i$
end
return $VOIestimate$

single task. Since the number of possible sequences of outcomes is exponential in the (single-task) horizon, it is not feasible for us to analytically compute the task-level VOI. Instead we approximate it by sampling.

Algorithm 2 shows our task-level VOI estimation, specifically for the case of acting in a finite set of MABs, though it would be straightforward to generalize this. We first sample a set of MAB parameters from the current multitask posterior b. We then simulate an exploratory run of RTBSS using the sampled parameters as if they were the true underlying MAB parameters. Exploration in this case could constitute, for example, pulling each arm 3 times, then running RTBSS until task horizon - we call the resulting variation ExploreSingleTask. After the simulation of the exploratory run is complete, the posterior b is updated to b'_{expl}. This is followed by simulating a non-exploratory RTBSS run that starts from the updated posterior b'_{expl}. At the end of these runs we obtain an estimate of reward we could get in the next two tasks in case the first one used additional exploration to refine multi-task posterior. We then compare the resulting reward to the case of not using additional exploration. Again starting from the current multitask posterior b, we simulate using RTBSS to act in one task, update multi-task posterior to b', then utilize b' for the second non-exploratory run. At the end of these non-exploratory runs we obtain an estimate of reward we could get in the next two tasks running RTBSS with no additional exploration.

The simulated runs allow us to compute an estimate of how much additional reward we can expect to gain if we use the next task for collecting additional information about the multi-task posterior. Notice that whenever our multi-task posterior reflects high uncertainty about the MAB parameters, we are likely to sample parameters (ϕ_i, θ_i) that would be "surprising" – could differ significantly from the most probable parameters suggested by the posterior. In such cases using additional exploration could be very informative, which would be reflected in higher rewards obtained in the RTBSS run following an exploratory run. Thus our VOI estimate would increase, indicating that more exploration during the next task would be warranted.

3.3 Approximations to Reduce Computation Time for Long-horizon Tasks

To speed up MT-VOI for tasks with long horizons we made several approximations.

When running single-task RTBSS we limited the depth of the tree. We found that limiting tree depth to 4 produced strategies that were as good as those obtained by trees with higher depths of up to 25. Limiting the depth of the search tree was critical to enable feasible run times for long-horizon tasks.

When using multi-task posterior for planning within a single task, we used a single component with mean closest to that of the mixture from each Dirichlet/Beta mixtures for each (MAB,arm). Even though after the first few tasks such approximation could be somewhat crude, its precision improves very quickly as the number of tasks completed grows (see more details on this in Appendix C.2).

We also avoided re-building the search tree in tasks with very long horizons unless there were non-negligible shifts in the multi-task posterior. When the posterior remains almost exactly the same (for example after pulling non-informative arm when enough evidence has already been collected for this arm), the search tree is unlikely to be altered enough to suggest a different optimal action.

4 Experiments

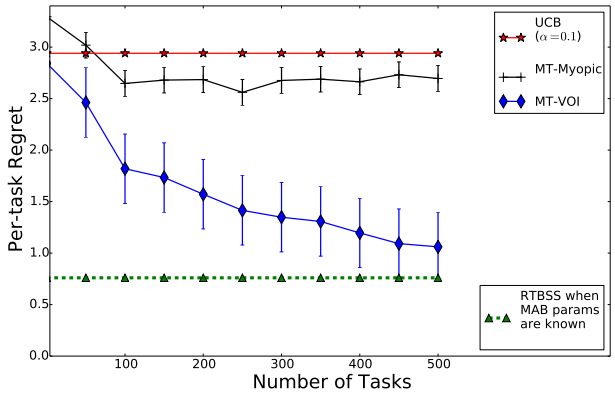
This section describes experiments for evaluating performance of our MT-VOI algorithm on two multi-task MAB domains. Our experiments demonstrate that MT-VOI can start from an uninformed prior over the multi-task parameters and achieve good performance. To our knowledge there are no established multi-task bandit benchmarks, therefore we discuss the performance on a domain with interesting properties introduced in the previous section and on a domain constructed in the most relevant prior work.

In our experiments we examine two groups of algorithms. The first group includes single task algorithms that either ignore (UCB) or leverage information given about the true parameters (RTBSS when MAB parameters are known). In terms of maximizing per-task reward, RTBSS with known MAB parameters should obtain the best per-task reward (or, equivalently, lowest per-task regret) we can hope for (up to error introduced by approximate planning). The second group contains algorithms that are initialized with uninformed prior – probability of encountering different MABs and their reward parameters are unknown. These algorithms either reflexively use information obtained in the process of completing tasks (MT-MYOPIC described below) or proactively try to optimize multi-task performance (MT-VOI).

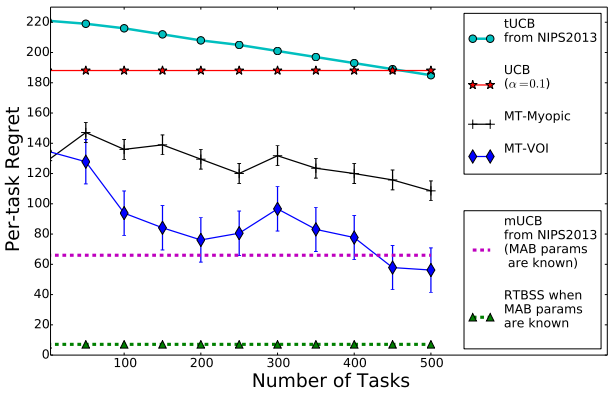
In order to perform single-task planning, while incorporating information gathered so far into the multi-task posterior, we employ RTBSS, using our analytic expressions to update the belief state of the POMDP modeled by RTBSS. We call the resulting algorithm MT-MYOPIC – when we perform RTBSS planning over a single task horizon and update the multi-task posterior at the end of each task. MT-MYOPIC offers an effective way of taking advantage of multi-task posterior for planning in a single task; however using it to plan in each task separately might cause us to under-explore multi-task posterior. We contrast the performance of MT-MYOPIC with MT-VOI (which uses task-level VOI estimate to determine whether the next task should involve more exploration).

For MT-VOI experiments we set the number of samples for estimating VOI (n_{smpl}) to 20. For

Figure 2: Comparing regret performance of MT-VOI in different domains
(plots smoothed by averaging over a window of size 50).



(a) InforArm domain with a discriminative arm.



(b) Domain with 5 MABs analyzed in NIPS2013 paper.

RTBSS with known MAB parameters we set the maximum depth to 25. When running RTBSS (as a part of MT-VOI and MT-MYOPIC) we experiment with maximum search tree depths from 4 to 25, and finally choose the maximum depth of 4 (since the strategies produced do not improve when increasing the maximum depth further).

4.1 Performance on InfoArm Domain

We first consider InfoArm domain introduced in "Bayesian Multi-task MAB Learning" section (Table 2). Figure 2a shows a comparison of per-task regret (averaged over 100 runs) when completing 500 tasks, with a task horizon of 100 actions in each task.

For establishing a baseline we implemented UCB [2] – a popular optimism under uncertainty single-task MAB algorithm that does not make use of prior information. We ran UCB with several parameters and picked the version that performed best on the InfoArm domain. We also implemented and ran MT-MYOPIC, MT-VOI and RTBSS with known MAB parameters.

MT-MYOPIC initialized with uninformed prior performed as well as UCB with best parameters, however even after completing 500 tasks it was not able to get close to the performance of RTBSS with known MAB parameters. In contrast, MT-VOI initialized with uninformed prior exhibited a much faster drop in per-task regret and eventually got close to the performance of RTBSS with known MAB parameters. This signified that additional targeted exploration was helpful. When our VOI estimate indicated that additional exploration was needed on the next task, we were able to refine the posterior using this additional exploration (we chose a rather conservative exploration strategy of doing 3 exploratory pulls for each arm before running RTBSS for the remaining part of the task).

4.2 Performance on NIPS2013 Tasks

We now consider the domain described by Azar et al. [3]. This domain contains 5 MABs with 7 arms each, and a horizon of 5000 steps. Each MAB_i has arm_i as its optimal arm. This domain

is challenging as the payoff for arm_3 is exactly the same for all the MABs (thus providing no information), and multiple MABs have very similar parameters across the arms.

	arm_1	arm_2	arm_3	arm_4	arm_5	arm_6	arm_7
MAB_1	0.9	0.75	0.45	0.55	0.58	0.61	0.65
MAB_2	0.75	0.89	0.45	0.55	0.58	0.61	0.65
MAB_3	0.2	0.23	0.45	0.35	0.3	0.18	0.25
MAB_4	0.34	0.31	0.45	0.725	0.33	0.37	0.47
MAB_5	0.6	0.5	0.45	0.35	0.95	0.9	0.8

Table 2: Rewards for arms in NIPS2013 domain in [3]

We used UCB with best-performing parameters to establish a baseline. In addition, we compared our performance to tUCB and mUCB – algorithms introduced at NIPS2013 by [3]. As described by the authors of NIPS2013 paper, tUCB used rewards for learning MAB parameters, while mUCB was provided with MAB parameters and only had to resolve the identity of the MAB. Figure 2b reveals that MT-MYOPIC achieves lower average per-task regret than UCB and tUCB. MT-VOI exhibits an even stronger performance. After ≈450 tasks it achieves average per-task regret of less than 60 – lower than mUCB (which assumes that MAB parameters are known and only needs to resolve MAB identity in each task). In contrast to MT-VOI, tUCB achieves regret below 100 only after 1500 tasks, not reaching the performance of mUCB even after 5000 tasks (these tUCB numbers are off the chart in Figure 2b, but can be seen in Figure 8 of [3]).

5 Conclusion and Future Work

In this work we introduced MT-VOI, an algorithm for optimizing performance across a series of tasks. Our results suggest a substantial improvement can be attained by explicitly reasoning about tasks to come. As discussed in the experiments section, our multi-task planning approach based on reasoning about value of information outperformed UCB, planning myopically, and multi-task planning algorithms from [3]. This suggests that explicitly reasoning about exploration/exploitation across multiple tasks is beneficial when optimizing multi-task rewards. We have also demonstrated that MT-VOI is able to minimize per-task regret effectively even when starting from uninformed prior. MT-VOI learned both the posterior for the distribution of MABs and the parameters of each MAB. Moreover, the algorithm was well-suited to refining the posterior only for the arms that showed promise in each MAB without the need to further refine the posterior for the arms that were far from the optimal ones.

Our MT-VOI algorithm is an approximate solution to maximizing the expected reward across multiple tasks. While we demonstrated that can outperform tUCB and mUCB from [3], the theoretical strength of these approaches is the availability of the proof for the regret bounds (which could be a significant improvement over standard single-task bounds which would be $O(J \log H)$). Due to the approximations performed by MT-VOI (discussed in exp

Though it is unclear if we can obtain regret bounds for our current algorithm, it is certainly interesting to derive regret bounds for approaches that seek to maximize multi-task performance directly.

Though in this paper we focused on acting in domains sampled from a finite sequence of multi-armed bandits, our MT-VOI algorithm could be generically applied to multi-task planning across a series of sequential decision making under uncertainty tasks, assuming each task is sampled from a stationary (if unknown) distribution. So this could be one promising one direction for future work.

Another direction would be to explore whether the Monte Carlo Tree Search (MCTS) [4] could be applied to optimizing multi-task rewards and compare to MT-VOI.

might be beneficial in this setting. We did investigate using the MCTS method UCT [9] instead of depth-limited RTBSS for our single-task internal planner, but found no benefit of UCT over RTBSS. This is likely because in the bandit setting we are provided with a reward signal at each step, whereas UCT and MCTS methods can be particularly beneficial given delayed rewards. It remains an open question if MCTS could be used to directly optimize multi-task action selection, though the horizon length involved seems challenging.

6 Bibliography

[1] John Asmuth and Michael L. Littman. Learning is Planning: Near Bayes-optimal Reinforcement Learning via Monte-Carlo Tree Search. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 19–26, 2011. 2

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. 4.1

[3] Mohammad Azar, Alessandro Lazaric, and Emma Brunskill. Sequential Transfer in Multi-armed Bandits with Finite Set of Models. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2013. 1.2, 1.3, 4.2, 2, 4.2, 5

[4] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, and et al. A Survey of Monte Carlo Tree Search Methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI*, 2012. 5

[5] Emma Brunskill, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Planning in Partially-observable Switching-mode Continuous Domains. *Annals of Mathematics and Artificial Intelligence*, 58(3-4):185–216, April 2010. 2.5

[6] Emma Brunskill and Lihong Li. Sample Complexity of Multi-task Reinforcement Learning. In *Proc. of the Twenty-Ninth Conf. on Uncertainty in Artificial Intelligence*, pages 122–131, 2013. 1.2

[7] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In *AIIDE*, 2008. 3.1

[8] M.O. Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002. 2

[9] Levente Kocsis and Csaba Szepesvri. Bandit based Monte-Carlo Planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006. 5

[10] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. 2.1, 2.3, A.1, A.3

[11] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. 3.1

[12] Alessandro Lazaric and Marcello Restelli. Transfer from Multiple MDPs. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2012. 1.2

[13] Sebastien Paquet. *Distributed Decision-making and Task Coordination in Dynamic, Uncertain and Real-time Multiagent Environments*. PhD thesis, University Laval, Quebec, Canada, 2006. 3.1

[14] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs, 2003. 2.5

[15] Pascal Poupart, Nikos A. Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 697–704, 2006. 2

[16] Stéphane Ross, Joelle Pineau, Brahim Chaib-draa, and Pierre Kreitmann. A Bayesian Approach for Learning and Planning in Partially Observable Markov Decision Processes. *J. Mach. Learn. Res.*, 12:1729–1770, July 2011. 2.5

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. 3.1

[18] David Silver and Joel Veness. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172, 2010. 3.1

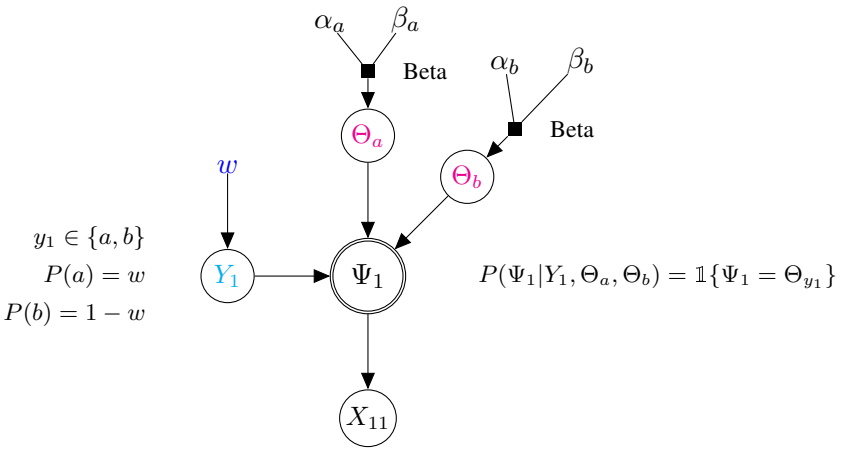
[19] Matthew E. Taylor and Peter Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009. 1.2

[20] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach. In *Proc. of the Twenty-Fourth Int'l Conf. on Machine Learning*, pages 1015–1022, 2007. 1.2, 2.2

A From Ground Bayesian Network to Generalized Plate Model

A.1 Growing the Ground Network

Let us start building Ground Bayesian Network (see Figure 17.7 in [10] for a related example). We will use notation introduced in section 2.1 for random variables composing the network (so please read/review section 2.1 before proceeding). For now let's suppose we start acting in the first task, and sample one of two possible 1-armed MABs $\{a, b\}$. We start with trivial 1-armed bandits in order to check that our intuition agrees with the outputs of the model in this simplest cases (and we will soon generalize to n-armed MABs).



In the network above w is the prior on the probabilities of each of the MAB types (for now w is a constant; it will be replaced by a Dirichlet prior shortly). $y_1 \in \{a, b\}$ is the type of the first MAB we will be interacting with. Ψ_1 is a multiplexer which evaluates to 1 iff the input corresponds to Θ_{y_1}, e.g. $P(\Psi_1 = \theta_a|Y_1 = a, \Theta_a = \theta_a, \Theta_b = \theta_b) = 1$ while $P(\Psi_1 = \theta_a|Y_1 = b, \Theta_a = \theta_a, \Theta_b = \theta_b) = 0$. Θ_a is the distribution on the parameters of MAB_a for the first arm. Θ_b is the distribution on the parameters of MAB_b for the first arm. X_{11} is the reward obtained from the first action in the first task.

Let's compute $P(X_{11} = 1)$ and see that we get something close to what our intuition suggests.

$$P(X_{11}{=}1) = \sum_{y_1=a,b} \sum_{\psi} \int_{\theta_a} \int_{\theta_b} P(X_{11} = 1, Y_1, \Psi_1, \Theta_a, \Theta_b) d\theta_a d\theta_b$$

since our joint distribution factorizes over the graph of our network, we obtain:

$$= \sum_{y_1=a,b} \sum_{\psi} \int_{\theta_a} \int_{\theta_b} P(X_{11} = 1|\Psi_1) P(\Psi_1|Y_1, \Theta_a, \Theta_b) P(y_1|w) f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b$$

$$= P(Y_1{=}a|w) \int_{\theta_a} \int_{\theta_b} \theta_a f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b + P(Y_1{=}b|w) \int_{\theta_a} \int_{\theta_b} \theta_b f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b$$

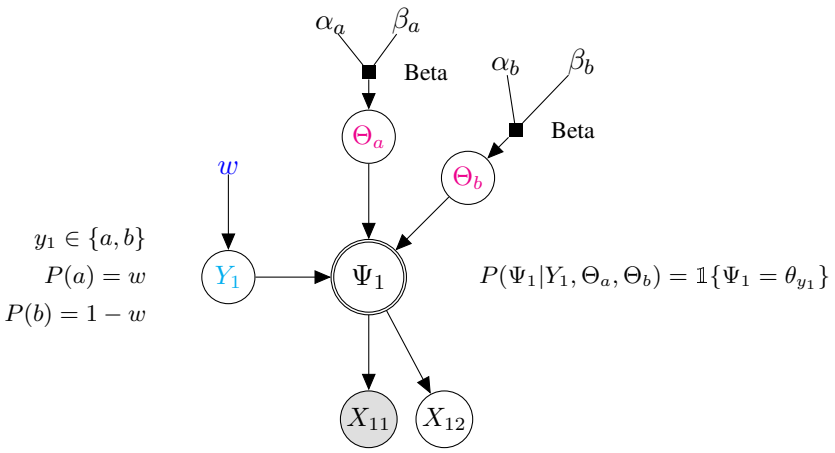
$$= P(Y_1{=}a|w) \int_{\theta_a} \theta_a f_{\Theta_a}(\theta_a) d\theta_a + P(Y_1{=}b|w) \int_{\theta_b} \theta_b f_{\Theta_b}(\theta_b) d\theta_b$$

$$= w E_{\Theta_a}[\Theta_a] + (1 - w) E_{\Theta_b}[\Theta_b]$$

This is exactly what we expected intuitively: the probability of getting a reward of 1 on our first pull is the expected value of the payoff parameter for MABs a and b weighted by the probability of sampling each of the MABs for our first task.

A.2 Bayesian Inference from Evidence in One Task

Suppose we observed $X_{11} = 1$ upon our first action and now we want to run inference on how likely we are to get a 1 from the next action, i.e. compute $P(X_{12}|X_{11})$. This situation would correspond to the ground Bayesian network below (note the shaded node for X_{11} indicating that it has been observed):



Since we want to express the fact that we are dealing with the same (though unknown) task, we will need to run the inference on the network in one step (see Appendix A.3 for a detailed explanation). We will start with the initial parameters for the network and compute the joint distribution of $P(Y_1, \Theta_a, \Theta_b, \Psi_1, X_{11}, X_{12})$, then marginalize it to get $P(X_{11}, X_{12})$, then using the definition of conditional probability obtain $P(X_{12}|X_{11}) = \frac{P(X_{12}, X_{11})}{P(X_{11})}$.

$$P(X_{12} = 1, X_{11} = 1) = \sum_{y_1=a,b} \sum_{\psi} \int_{\theta_a} \int_{\theta_b} P(X_{12}, X_{11}, Y_1, \Psi_1, \Theta_a, \Theta_b) d\theta_a d\theta_b$$

$$= \sum_{y_1=a,b} \sum_{\psi} \int_{\theta_a} \int_{\theta_b} P(X_{12}, X_{11}|Y_1, \Psi_1, \Theta_a, \Theta_b) f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) P(y_1|w) d\theta_a d\theta_b$$

notice that $P(X_{12}, X_{11}|Y_1, \Psi_1, \Theta_a, \Theta_b) = P(X_{12}|Y_1, \Psi_1, \Theta_a, \Theta_b) P(X_{11}|Y_1, \Psi_1, \Theta_a, \Theta_b)$, since X_{12}, X_{11} are independent given parents:

$$= \sum_{y_1=a,b} P(y_1|w) \int_{\theta_a} \int_{\theta_b} P(X_{12}|Y_1, \Psi_1, \Theta_a, \Theta_b) P(X_{11}|Y_1, \Psi_1, \Theta_a, \Theta_b) f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b$$

$$= w \int_{\theta_a} \int_{\theta_b} P(X_{12}|\Theta_a) P(X_{11}|\Theta_a) f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b$$

$$+ (1 - w) \int_{\theta_a} \int_{\theta_b} P(X_{12}|\Theta_b) P(X_{11}|\Theta_b) f_{\Theta_a}(\theta_a) f_{\Theta_b}(\theta_b) d\theta_a d\theta_b$$

$$= w \int_{\theta_a} \Theta_a \cdot \Theta_a f_{\Theta_a}(\theta_a) d\theta_a + (1-w) \int_{\theta_b} \Theta_b \cdot \Theta_b f_{\Theta_b}(\theta_b) d\theta_a$$
$$= w\ E_{\Theta_a}[\Theta_a^2] + (1-w) E_{\Theta_b}[\Theta_b^2]$$

$$P(X_{12}=1|X_{11}=1) = \frac{P(X_{12}=1, X_{11}=1)}{P(X_{11}=1)} = \frac{w\ E_{\Theta_a}[\Theta_a^2] + (1-w) E_{\theta_b}[\Theta_b^2]}{P(X_{11}=1)} \tag{13}$$

More generally, if we made r observations with reward 1 and z observations with reward 0, the probability of getting a reward of 1 on the next observation would be:

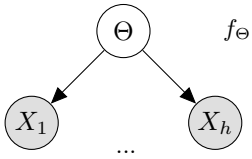
$$P(\text{reward}=1|r,z) = \frac{P(r=1,r,z)}{P(r,z)} = \frac{w\ E[\Theta_a^{r+1}(1-\Theta_a)^z] + (1-w)E[\Theta_b^{r+1}(1-\Theta_b)^z]}{w\ E[\Theta_a^{r}(1-\Theta_a)^z] + (1-w)E[\Theta_b^{r}(1-\Theta_b)^z]}$$

Hence we see that inference in a single task involves special higher moments of Beta-distributed variables (see Appendix B). For generalization of inference with multi-armed MABs and Dirichlet prior instead of the constant weight w see section 2.3.

A.3 Alternative Approaches to Inference in One Task

Let's consider other ways we could compute $P(X_{12}=1|X_{11}=1)$. Initially it seems that it should be possible to compute updated conditional probability distributions (CPDs) $f_{\Theta_a}(\Theta_a|X_{11}=1), f_{\Theta_b}(\Theta_b|X_{11}=1), P(Y_1|w, X_{11}=1)$, do Bayesian update replacing old CPDs by the new ones and then compute $P(X_{12}=1)$. However, since variable Y_1 is hidden (never observed) we will have an activated v-structure in the graph between Y_1, Θ_a, Θ_b variables when some rewards from the task are observed (see section 3.3.1 of [10] for details on v-structure activation). Hence, if updates within the same task are executed sequentially, X_{11} and X_{12} would become dependent. As a result Θ parameters would become dependent and inference would become problematic. Therefore we will not model the arrival of new observations in the same task by adding new reward nodes for each observation sequentially. Instead, we will do batch updates incorporating all the observations from the same task in one update. In practice we will need to run inference before we are done interacting with the task. We will do this by initializing our bayesian network to the state it was in right before the interaction with the current task has begun, and then add nodes to represent all the evidence obtained in the current task so far (at once). We then run inference to predict the probability of obtaining 1 for the next reward from this task.

Batch update is not the only way to run inference in one task, there is also an alternative approach, which considers a mixture of MAB parameters weighted by MAB probabilities cut out of the larger Bayesian Network structure. Consider the following very simple Bayesian network:



$$f_\Theta(\theta) = w f_{Beta(\alpha_a,\beta_a)}(\theta) + (1-w) f_{Beta(\alpha_b,\beta_b)}(\theta)$$

(where $f_{Beta(\alpha,\beta)}$ denotes PDF of Beta-distributed random variable with parameters α, β)

If we observe $X_1 = 1$, we can perform Bayesian update for Θ as follows:

$$f_\Theta(\Theta|X_1=1) = \frac{P(X_1=1|\Theta) f_\Theta(\theta)}{P(X_1=1)}$$

$$P(X_1{=}1|\Theta{=}\theta)f_\Theta(\Theta{=}\theta) = \theta\left(w\frac{\Gamma(\alpha_a{+}\beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)}\theta^{\alpha_a\text{-}1}(1{-}\theta)^{\beta_a\text{-}1} + (1{-}w)\frac{\Gamma(\alpha_b{+}\beta_b)}{\Gamma(\alpha_b)\Gamma(\beta_b)}\theta^{\alpha_b\text{-}1}(1{-}\theta)^{\beta_b\text{-}1}\right)$$

$$= w\,\frac{\Gamma(\alpha_a{+}\beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)}\frac{\Gamma(\alpha_a{+}1)\Gamma(\beta_a)}{\Gamma(\alpha_a{+}\beta_a{+}1)}f_{Beta(\alpha_a+1,\beta_a)}(\theta)$$
$$+ (1{-}w)\frac{\Gamma(\alpha_b{+}\beta_b)}{\Gamma(\alpha_b)\Gamma(\beta_b)}\frac{\Gamma(\alpha_a{+}1)\Gamma(\beta_b)}{\Gamma(\alpha_b{+}\beta_b{+}1)}f_{Beta(\alpha_b+1,\beta_b)}(\theta)$$
$$= w\,\frac{\alpha_a}{\alpha_a{+}\beta_a}f_{Beta(\alpha_a+1,\beta_a)}(\theta) + (1{-}w)\frac{\alpha_b}{\alpha_b{+}\beta_b}f_{Beta(\alpha_b+1,\beta_b)}(\theta)$$

$$P(X_1{=}1){=}\int_\theta P(X_1=1,\Theta)d\theta = w\,\frac{\alpha_a}{\alpha_a{+}\beta_a} + (1{-}w)\frac{\alpha_b}{\alpha_b{+}\beta_b}$$

$$f_{\Theta_{new_1}}(\theta) \leftarrow f_\Theta(\Theta|X_1{=}1) = \frac{w\,\frac{\alpha_a}{\alpha_a+\beta_a}f_{Beta(\alpha_a+1,\beta_a)} + (1{-}w)\frac{\alpha_b}{\alpha_b+\beta_b}f_{Beta(\alpha_b+1,\beta_b)}}{P(X_1{=}1)}$$

We can now compute $P(X_2 = 1)$ using the updated $f_{\Theta_{new_1}}$:

$$P(X_2{=}1){=}\int_\theta P(X_2{=}1|\Theta_{new_1})f_{\Theta_{new_1}}(\theta)d\theta$$
$$= \frac{1}{P(X_1{=}1)}\left(w\,\frac{\alpha_a}{\alpha_a{+}\beta_a}\frac{\alpha_a{+}1}{\alpha_a{+}\beta_a{+}1} + (1{-}w)\frac{\alpha_b}{\alpha_b{+}\beta_b}\frac{\alpha_b{+}1}{\alpha_b{+}\beta_b{+}1}\right)$$
$$= \frac{w\,E_{\Theta_{a_{old}}}[\Theta^2_{a_{old}}] + (1{-}w)E_{\Theta_{b_{old}}}[\Theta^2_{b_{old}}]}{P(X_1{=}1)},\ \text{where } \Theta_{a_{old}} \sim Beta(\alpha_a,\beta_a), \Theta_{b_{old}} \sim Beta(\alpha_b,\beta_b)$$

We can clearly see that we obtain the same pattern as when performing batch updates in the full Bayesian network (equation 13), so this approach of cutting out a restricted piece of the network and running incremental Bayesian updates for the same task is consistent with the results we obtained before. If we were to incorporate the second observation into Θ we would obtain:

$$f_{\Theta_{new_2}}(\theta) \leftarrow f_{\Theta_{new_1}}(\Theta_{new_1}|X_2{=}1)$$
$$= \frac{w\,\frac{\alpha_a}{\alpha_a+\beta_a}\frac{\alpha_a+1}{\alpha_a+\beta_a+1}f_{Beta(\alpha_a+2,\beta_a)}(\theta) + (1{-}w)\frac{\alpha_b}{\alpha_b+\beta_b}\frac{\alpha_b+1}{\alpha_b+\beta_b+1}f_{Beta(\alpha_b+2,\beta_b)}(\theta)}{w\,\frac{\alpha_a}{\alpha_a+\beta_a}\frac{\alpha_a+1}{\alpha_a+\beta_a+1} + (1{-}w)\frac{\alpha_b}{\alpha_b+\beta_b}\frac{\alpha_b+1}{\alpha_b+\beta_b+1}} \tag{14}$$

Let us compare the two approaches we have used so far (batch and cut-out) with the approach of batch updates in the cut-out piece of the network.

$$f_\Theta(\Theta|X_1=1,X_2=1) = \frac{P(X_1=1,X_2=1|\theta)f_\Theta(\Theta)}{P(X_1=1,X_2=1)}$$
$$P(X_1=1,X_2=1|\theta)f_\Theta(\Theta) = P(X_1=1|\theta)P(X_2=1|\theta)f_\Theta(\Theta) = \theta^2 f_\Theta(\Theta)$$
$$= \theta^2\left(w\,\frac{\Gamma(\alpha_a{+}\beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)}\theta_a^\alpha(1-\theta)^{\beta_a} + (1{-}w)\frac{\Gamma(\alpha_b{+}\beta_b)}{\Gamma(\alpha_b)\Gamma(\beta_b)}\theta_b^\alpha(1-\theta)^{\beta_b}\right)$$
$$= w\,\frac{\Gamma(\alpha_a{+}\beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)}\theta^{\alpha_a+2}(1-\theta)^{\beta_a} + (1{-}w)\frac{\Gamma(\alpha_b{+}\beta_b)}{\Gamma(\alpha_b)\Gamma(\beta_b)}\theta^{\alpha_b+2}(1-\theta)^{\beta_b}$$
$$= w\,\frac{\Gamma(\alpha_a{+}\beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)}\frac{\Gamma(\alpha_a{+}2)\Gamma(\beta_a)}{\Gamma(\alpha_a{+}\beta_a{+}2)}f_{Beta(\alpha_a+2,\beta_a)}(\theta)$$

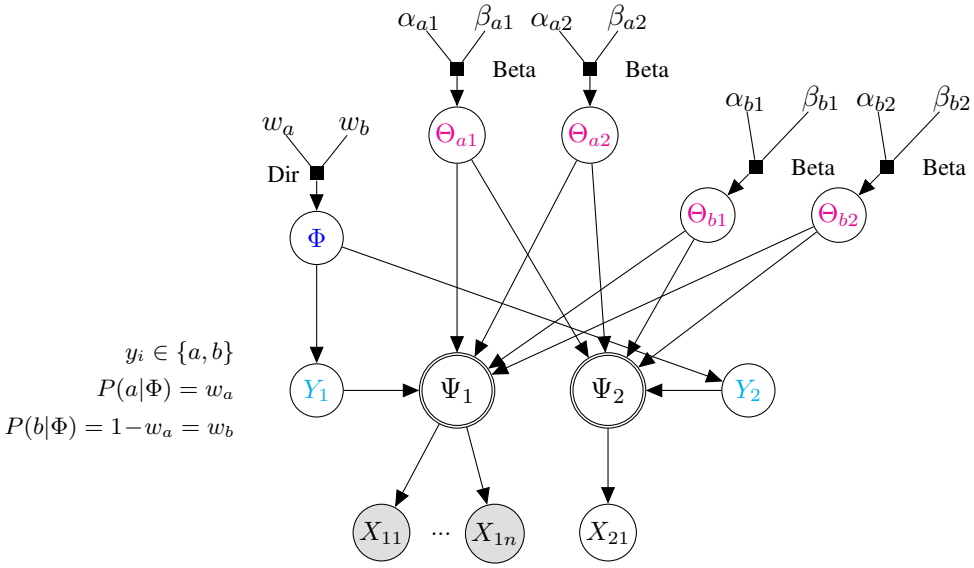
$$+ (1-w)\frac{\Gamma(\alpha_b+\beta_b)}{\Gamma(\alpha_b)\Gamma(\beta_b)}\frac{\Gamma(\alpha_b+2)\Gamma(\beta_b)}{\Gamma(\alpha_b+\beta_b+2)} f_{Beta(\alpha_b+2,\beta_b)}(\theta)$$

$$= w\,\frac{\alpha_a}{\alpha_a+\beta_a}\frac{\alpha_a+1}{\alpha_a+\beta_a+1} f_{Beta(\alpha_a+2,\beta_a)}(\theta) + (1-w)\frac{\alpha_b}{\alpha_b+\beta_b}\frac{\alpha_b+1}{\alpha_b+\beta_b+1} f_{Beta(\alpha_b+2,\beta_b)}(\theta)$$

Since $P(X_2{=}1, X_1{=}1) = w\frac{\alpha_a}{\alpha_a+\beta_a}\frac{\alpha_a+1}{\alpha_a+\beta_a+1} + (1-w)\frac{\alpha_b}{\alpha_b+\beta_b}\frac{\alpha_b+1}{\alpha_b+\beta_b+1}$, we obtain that $f_\Theta(\Theta|X_1{=}1, X_2{=}1)$ is equivalent to equation 14 and we again get the same result with this third way of incorporating evidence.

A.4 From Ground Network for Several Tasks to Plate Model

Let us now expand our ground Bayesian network to include several tasks. In the illustration below we also replace constant weight w with a Dirichlet-distributed random variable Φ (to capture our beliefs about probabilities of encountering each MAB) and expand each of the MABs to have two arms.



Now the need for a more compact notation is obvious, so we collect all the MABs into a single plate (of size M - the total number of MABs) that is indexed by MAB identity, and add a nested plate of size N for each of the arms. Likewise we collect all the Y_js into a plate of size J (the total number of tasks) with a nested plate for each selector $\Psi_{j,n}$ and a further nested plate for observations X (indexed by task number, arm and step number within the task). The result is the plate model presented in section 2.1 (Figure 1).

Note that there is an alternative way of thinking about our Bayesian network after completing a task. When we are done interacting with the previous task we can update:

$$f_{\Theta_{m,n}}(\theta) \leftarrow f_{\Theta_{m,n}}(\Theta_{m,n}|\text{rewards from finished task}),\ \text{for } m = 1...M, n = 1...N$$

$$f_\Phi(\phi) \leftarrow f_\Phi(\Phi|\text{rewards from finished task})$$

Thus instead of expanding the ground Bayesian network further, we will expand the sizes of Beta and Dirichlet mixtures for Φ and $\Theta_{m,n}$ nodes. For the purposes of inference (needed for planning)

these two views are identical. If we expand the ground network upon arrival of new observations and tasks, when performing inference we will obtain the same Beta and Dirichlet mixtures. So we can either incorporate evidence from the completed task into a Bayesian update (expansion of the mixtures), or view the ground network for the plate model in Figure 1 as a network with nodes for the whole sequence of J tasks (with rewards already obtained represented by observed $X_{j,n,h}$ nodes and future rewards represented by unobserved nodes).

B Special Higher Moments of Beta Distribution

In probability theory higher moments of a random variable Θ are usually denoted by $E_\Theta[\Theta^r], r \in \mathbb{N}$. In this work we make frequent use of moments of the form $E_\Theta[\Theta^r(1-\Theta)^z], r, z \in \mathbb{N}$. Below we derive expressions for these special higher moments when $\Theta \sim Beta(\alpha, \beta)$.

$$E_\Theta[\Theta^r(1-\Theta)^z] = \int_\theta \theta^r(1-\theta)^z f_\Theta(\theta)d\theta = \int_0^1 \theta^r(1-\theta)^z \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}d\theta$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{r+\alpha-1}(1-\theta)^{z+\beta-1}d\theta$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot \frac{\Gamma(\alpha+r)\Gamma(\beta+z)}{\Gamma(\alpha+\beta+r+z)} \int_0^1 \frac{\Gamma(\alpha+\beta+r+z)}{\Gamma(\alpha+r)\Gamma(\beta+z)}\theta^{r+\alpha-1}(1-\theta)^{z+\beta-1}d\theta$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot \frac{\Gamma(\alpha+r)\Gamma(\beta+z)}{\Gamma(\alpha+\beta+r+z)}, \text{using } \Gamma(a) = (a-1)\Gamma(a-1) \text{ we get:}$$

$$= \frac{\Gamma(\alpha+\beta)\cdot(\alpha+r-1)(\alpha+r-2)\cdots\alpha\Gamma(\alpha)\cdot(\beta+z-1)(\beta+z-2)\cdots\beta\Gamma(\beta)}{\Gamma(\alpha)\Gamma(\beta)\cdot(\alpha+\beta+r+z-1)(\alpha+\beta+r+z-2)\cdots(\alpha+\beta)\Gamma(\alpha+\beta)}$$

$$= \frac{(\alpha+r-1)(\alpha+r-2)\cdots\alpha\cdot(\beta+z-1)(\beta+z-2)\cdots\beta}{(\alpha+\beta+r+z-1)(\alpha+\beta+r+z-2)\cdots(\alpha+\beta)}$$

$$= \frac{\prod_{i=0}^{r-1}(\alpha+i)\cdot\prod_{i=0}^{z-1}(\beta+i)}{\prod_{i=0}^{r+z-1}(\alpha+\beta+i)}$$

Or alternatively, using $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ we get $E_\Theta[\Theta^r(1-\Theta)^z] = \frac{B(\alpha+r,\beta+z)}{B(\alpha,\beta)}$, where B is Beta function $B(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt$.

Even though obtaining $E_\Theta[\Theta^r(1-\Theta)^z]$ looks computationally intensive, in our inference and Bayesian update implementation we never need to compute $E_\Theta[\Theta^r(1-\Theta)^z]$ from scratch for large r, z. Instead most cases require computing $E_\Theta[\Theta^{r+1}(1-\Theta)^z]$ or $E_\Theta[\Theta^r(1-\Theta)^{z+1}]$, when $E_\Theta[\Theta^r(1-\Theta)^z]$ has already been obtained. And so we use a very fast update rule (the case for $z+1$ is similar):

$$E_\Theta[\Theta^{r+1}(1-\Theta)^z] = (\alpha+r)\cdot E_\Theta[\Theta^r(1-\Theta)^z]$$

C Tackling Beta and Dirichlet Mixture Expansion

C.1 Growth Rate of Beta and Dirichlet Mixtures due to Bayesian Updates

As we mentioned in Appendix A.4, we can incorporate evidence from the completed tasks into a Bayesian update for $\Theta_{m,n}$ and Φ nodes. We presented derivations for these Bayesian updates in section 2.4 (equations 9 and 12), and mentioned the growth of Beta and Dirichlet mixtures.

From equation 9 we can see that after each update the size of the mixture of Betas for each $\Theta_{m,n}$ doubles. The first part of the sum $w_{\Phi_m} w_{\mu_{\Theta_{m \setminus n}}} \sum_{k=1}^{K} w_k f_{\mathcal{B}_k^+}(\theta)$ contributes K new beta components to the mixture, while the second part $\left(\sum_{\substack{y=1, \\ y \neq m}}^{M} w_{\Phi_y} w_{\mu_{\Theta_y}} \right) f_{\Theta_{m,n}}(\theta)$ only changes weights of the K already existing Beta-distributed components.

As to Bayesian updates for Φ – at the first glance at equation 12 we could conclude that because of the double summation $\sum_{y=1}^{M} \sum_{k=1}^{K}$ the mixture $f_\Phi(\phi)$ grows exponentially with the number of tasks completed (M^j). However we note that most of the new components in the mixture could be grouped with already existing ones. Let Φ_j denote the updated mixture after completing j^{th} task, then we have:

$$
\begin{aligned}
&\Phi_0 \sim Dir(\alpha_1, ..., \alpha_M) \\
&\Phi_1 \sim Dir(\alpha_1{+}1, ..., \alpha_M){+}...{+}Dir(\alpha_1, ..., \alpha_M{+}1) \\
&\Phi_2 \sim Dir(\alpha_1{+}2, ..., \alpha_M){+}...{+}Dir(\alpha_1{+}1, ..., \alpha_M{+}1) + ... \\
&\quad + Dir(\alpha_1, ..., \alpha_i{+}1, ..., \alpha_M{+}1){+}...{+}Dir(\alpha_1, ..., \alpha_i{+}1, ...\alpha_M{+}2) + ... \\
&\quad + Dir(\alpha_1{+}1, ..., \alpha_M{+}1){+}...{+}Dir(\alpha_1, ..., \alpha_M{+}2)
\end{aligned}
$$

We observe that the number of unique Dirichlet components in a mixture Φ_j is equal to the number of ways to choose one of M places j times with replacement. Each time we would choose a place $m = 1...M$ we would increment the corresponding parameter of the Dirichlet component, but for the final result the order in which we increment does not matter. Thus we can group all of the Dirichlet-distributed components of the Φ_j mixture into $\binom{M+j-1}{j-1}$ components (combinatorial formula for choosing j times from M objects with replacement, order does not matter). As j increases, the number of components in the mixture grows as $\frac{(M+j-1)!}{(j-1)!M!} = \frac{1}{M!} \cdot j(j+1) \cdots (j+M-1) = O(j^M)$. So the Dirichlet mixture Φ would grow polynomially with the number of tasks completed (and not exponentially as it might first appear).

C.2 Using Collapsed Posterior to Speed up Inference

Since the number of Beta components for each $\Theta_{m,n}$ and Φ grows with the number of tasks when using analytic Bayesian updates, we need to choose an effective approximation strategy. In section 2.4 we discussed several approaches to reducing mixture sizes. Here we describe and additional approach to speeding up inference during planning. This approach involves using a "collapsed" posterior for inference – using a single Beta/Dirichlet distribution to approximate a mixture of Betas/Dirichlets.

As to the experiments we ran so far, our use of the collapsed posterior most likely did not hurt the performance after we have collected enough samples to learn a fairly precise posterior of each $\Theta_{m,n}$. Which, for the NIPS task happened after around 15-25 tasks, since each of our task

horizons have been quite long (5K steps). It is worth noting that around that same point (after 15-25 steps) our exploration heuristic would also suggest to stop exploring, and inspection of a number of posteriors confirmed that by that time our estimate of $P(\text{reward=1})$ was usually quite precise even when using collapsed posterior. Hence despite the fact that analytically using the collapsed posterior is not equivalent to using the full multi-task posterior, this was a fair approach for the particular algorithm we have used for the NIPS task.

The main question we are going to tackle now is what is the difference between using the full multi-task posterior computed using the Bayesian updates (after observing rewards in the first $j-1$ tasks) and using a collapsed multi-task posterior to select actions in the j^{th} task. Collapsed posterior is obtained as follows:

- for each $\Theta_{m,n}$ mixture use a single Beta with parameters α, β such that $\frac{\alpha}{\alpha+\beta} = E_{\Theta_{m,n}}[\Theta_{m,n}]$, while $\alpha+\beta$ is equal to the total number of pulls of arm n completed before starting task j
- for mixture of Dirichlets Φ use a single Dirichlet with parameters $\alpha_1, ...\alpha_M$ such that $\frac{\alpha_i}{\sum_{i=1}^{M}\alpha_i} = E_{\Phi}[\Phi_i]\ \forall i = 1...M$ and $\sum_{i=1}^{M}\alpha_i$ is the number of tasks completed so far

The question we need to answer: what is the difference in probability reward=1 for arm n for the first step in task j computed when using multi-task posterior versus computed using collapsed posterior. For simplicity let us first work with a trivial planning algorithm – simply pick the best action given the current belief state $\boldsymbol{b}$ (updated multi-task posterior) and compute the expected reward we would receive if we pulled that arm until the end of the current task. Finding such action is useful for computing lower bound at the leaves (when using a search tree for planning).

$$m_{leaf} = \underset{m\in 1,...,M}{\operatorname{argmax}} P(Y_j = m|\boldsymbol{b}) = \underset{m\in 1,...,M}{\operatorname{argmax}} E_{\Phi}[\Phi_m]$$

$$n_{leaf} = \underset{a\in 1,...,N}{\operatorname{argmax}} P(\text{reward=1 from arm } n|m_{leaf}, \boldsymbol{b}) = \underset{n\in 1,...,N}{\operatorname{argmax}} E_{\Theta_{m_{leaf},n}}[\Theta_{m_{leaf},n}]$$

So far the two outcomes using full and collapsed posteriors would be identical, since we have only used the expectations of the Beta and Dirichlet mixtures. Let us look closely at what happens when we obtain another reward from the same arm in the same task. Recall that to compute the probability of reward given a certain MAB and arm we can use equation 6 from section 2.2.

$$P(\text{reward of 1 from arm } n|r_n=1, z_n=0, Y_j=m) = \frac{E_{\Theta_{m,n}}[\Theta^2_{m,n}]}{E_{\Theta_{m,n}}[\Theta_{m,n}]} \tag{15}$$

Let $\Theta \sim w_1 Beta(\alpha_1,\beta_1) + w_2 Beta(\alpha_2,\beta_2)$. Then using derivations in Appendix B we get:

$$E_{\Theta}[\Theta^r(1-\Theta)^z] = \sum_{k=1}^{K} w_k \frac{\prod_{i=0}^{r-1}(\alpha_k+i)\cdot\prod_{i=0}^{z-1}(\beta_k+i)}{\prod_{i=0}^{r+z-1}(\alpha_k+\beta_k+i)}$$

$$P(\text{reward of 1 from arm } n|r_n=1, z_n=0, Y_j=m) =$$

$$= \sum_{k=1}^{K} w_k \frac{\prod_{i=0}^{1}(\alpha_k+i)}{\prod_{i=0}^{1}(\alpha_k+\beta_k+i)} \Bigg/ \sum_{k=1}^{K} w_k \frac{\prod_{i=0}^{0}(\alpha_k+i)}{\prod_{i=0}^{0}(\alpha_k+\beta_k+i)} = \frac{\sum_{k=1}^{K} w_k \frac{(\alpha_k+1)\alpha_k}{(\alpha_k+\beta_k+1)(\alpha_k+\beta_k)}}{\sum_{k=1}^{K} w_k \frac{\alpha_k}{\alpha_k+\beta_k}}$$

$$P(\text{reward of 1 from arm } n|r_n=1, z_n=0) = \frac{\sum_{y=1}^{M} E_{\Phi}[\Phi_y] \sum_{k=1}^{K} w_k \frac{(\alpha_k+1)\alpha_k}{(\alpha_k+\beta_k+1)(\alpha_k+\beta_k)}}{\sum_{y=1}^{M} E_{\Phi}[\Phi_y] \sum_{k=1}^{K} w_k \frac{\alpha_k}{\alpha_k+\beta_k}}$$

We arrived at expressions that involve sums in the denominators, so it seems that it is not possible to make further simplifications. Hence the trouble comes when we take the next step (or when we expand the search tree). In particular, we will need to compute the probability of reward after having taken one step in the same task. For concreteness, let us suppose that we have pulled arm n and obtained a reward of 1 (or, equivalently, we are working with the corresponding child of the node in a search tree). In this situation we will need to evaluate equation 15, and thus work with $E_{\Theta_{m,n}}[\Theta^2_{m,n}]$. This is a second moment of a mixture of Betas, which will not in general be equal to the second moment of the single Beta in the collapsed posterior. To see this even more clearly, note that if we keep obtaining the reward of 1 until task horizon, we will need to keep computing higher and higher moments of the Beta mixture (versus higher moments of the single Beta in the collapsed posterior). All these moments will not be equal for two different probability distributions.

We could argue that, after a sufficient number of tasks completed, a single Beta can approximate the true distribution for $\Theta_{m,n}$ well. However, we do not have a reason to believe that this would happen after only the first few tasks (when the number of samples is small). Let's consider a tiny example. Let $\Theta_{m,n} = 0.1Beta(1,1)+0.9Beta(100,100)$. Then $E_{\Theta_{m,n}}[\Theta_{m,n}] = 0.05+0.45 = 0.5$. Let the collapsed posterior contain $\Theta_{m,n_{\text{collapsed}}} = Beta(101,101)$. Then, using the formulas for higher moments of Beta from Appendix B, we have:

$$E[\Theta^2_{m,n}] = 0.1 \cdot \frac{1}{2} \cdot \frac{2}{3} + 0.9 \cdot \frac{1}{2} \cdot \frac{101}{201} \approx 0.0333 + 0.2261 = 0.2594$$

$$E[\Theta^2_{m,n_{\text{collapsed}}}] = \frac{101}{202} \cdot \frac{102}{203} \approx 0.2512$$

$$E[\Theta^3_{m,n}] = 0.1 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} + 0.9 \cdot \frac{1}{2} \cdot \frac{101}{201} \cdot \frac{102}{202} \approx 0.1392$$

$$E[\Theta^3_{m,n_{\text{collapsed}}}] = \frac{101}{202} \cdot \frac{102}{203} \cdot \frac{103}{204} \approx 0.1268$$

And if we have a Beta mixture with a more even weight distribution $\Theta_{m,n} = 0.5Beta(1,1) + 0.5Beta(100,100)$, then $E[\Theta^2_{m,n}] \approx 0.2923, E[\Theta^3_{m,n}] \approx 0.1884$, which are farther from expectations for $\Theta_{m,n_{\text{collapsed}}}$.

D Multitask Planning Examples

D.1 InfoArm Domain Example

Below is a brief example that illustrates Bayesian belief state updates in a sequence of tasks from InfoArm domain (see description in table 2 and experiments in section 4.1).

Figures below show distributions for Φ (mixture of Dirichlets in tables with blue borders) and $\Theta_{m,n}$ (mixture of Betas in each cell of the tables with magenta borders). We initialize with uninformed prior:

1.00*Dir(5,5)

MAB means: [0.5000 0.5000] arms means: [0.5000 0.5000 0.5000] task MAB wts: [0.5000 0.5000]

unweighted means			
0.5000 0.5000 0.5000	1.00*Beta(1,1)	1.00*Beta(1,1)	1.00*Beta(1,1)
0.5000 0.5000 0.5000	1.00*Beta(1,1)	1.00*Beta(1,1)	1.00*Beta(1,1)

Then we start acting in a sequence of tasks. The updated posterior after the first task is finished:

[rewards: | arm=1 (r=2, z=1) | arm=2 (r=33, z=11) | arm=3 (r=48, z=5)]
Updated Posterior:

0.50*Dir(6,5)+0.50*Dir(5,6)

MAB means: [0.5000 0.5000] arms means: [0.5500 0.6196 0.6955] task MAB wts: [0.5000 0.5000]

unweighted means			
0.6000 0.7391 0.8909	1.00*Beta(3,2)	1.00*Beta(34,12)	1.00*Beta(49,6)
0.5000 0.5000 0.5000	1.00*Beta(1,1)	1.00*Beta(1,1)	1.00*Beta(1,1)

The updated multi-task posterior after the second task is finished:

[rewards: | arm=1 (r=3, z=0) | arm=2 (r=2, z=1) | arm=3 (r=85, z=9)]
Updated Posterior:

0.51*Dir(7,5)+0.45*Dir(6,6)+0.03*Dir(5,7)

MAB means: [0.5398 0.4602] arms means: [0.6989 0.6498 0.8063] task MAB wts: [0.5398 0.4602]

unweighted means			
0.7406 0.7350 0.8988	0.94*Beta(6,2) + 0.06*Beta(3,2)	0.94*Beta(36,13) + 0.06*Beta(34,12)	0.94*Beta(134,15) + 0.06*Beta(49,6)
0.6500 0.5500 0.6979	0.50*Beta(1,1) + 0.50*Beta(4,1)	0.50*Beta(1,1) + 0.50*Beta(3,2)	0.50*Beta(1,1) + 0.50*Beta(86,10)

After about 15 tasks are completed we see that the estimates of $P(\text{reward=1}|MAB=m, arm=n)$ (displayed under "unweighted means") come very close to the true probabilities. The updated multitask posterior after 16^{st} task is shown below:

[rewards: | arm=1 (r=91, z=1) | arm=2 (r=2, z=1) | arm=3 (r=3, z=2)]
Updated Posterior:

0.53*Dir(15,11)+0.44*Dir(14,12)+0.03*Dir(13,13)

MAB means: [0.5579 0.4421] arms means: [0.8526 0.8485 0.5237] task MAB wts: [0.5579 0.4421]

unweighted means			
0.9606 0.7426 0.8991	0.98*Beta(122,5) + 0.02*Beta(119,5)	0.94*Beta(49,17) + 0.06*Beta(47,16)	0.96*Beta(731,82) + 0.04*Beta(646,73)
0.7163 0.9820 0.0500	0.41*Beta(16,7) + 0.59*Beta(19,7)	0.83*Beta(553,10) + 0.17*Beta(555,11)	1.00*Beta(1,19)

Furthermore we can see that the learning algorithm discovered a strategy of pulling informative arm in order to distinguish between MABs faster.

Starting Task17
Collapsed posterior at task beginning
1.00*Dir(15,11)
MAB means: [0.5769 0.4231] arms means: [0.8634 0.8439 0.5399] task MAB wts: [0.5769 0.4231]
unweighted means
0.9606 0.7424 0.8991
0.7308 0.9822 0.0500

1.00*Beta(122,5)	1.00*Beta(49,17)	1.00*Beta(731,82)
1.00*Beta(19,7)	1.00*Beta(553,10)	1.00*Beta(1,19)

Took action 3 got reward 0 (step=0)
Took action 3 got reward 0 (step=1)
Took action 2 got reward 1 (step=2)
Took action 2 got reward 1 (step=3)
Took action 2 got reward 1 (step=4)
Took action 2 got reward 1 (step=5)
Took action 2 got reward 1 (step=6)

Starting Task18
Collapsed posterior at task beginning
1.00*Dir(15,12)
MAB means: [0.5556 0.4444] arms means: [0.8585 0.8482 0.5197] task MAB wts: [0.5556 0.4444]
unweighted means
0.9606 0.7424 0.8991
0.7308 0.9803 0.0455

1.00*Beta(122,5)	1.00*Beta(49,17)	1.00*Beta(731,82)
1.00*Beta(19,7)	1.00*Beta(648,13)	1.00*Beta(1,21)

Took action 3 got reward 1 (step=0)
Took action 3 got reward 1 (step=1)
Took action 1 got reward 1 (step=2)
Took action 1 got reward 1 (step=3)
Took action 1 got reward 1 (step=4)
Took action 1 got reward 1 (step=5)
Took action 1 got reward 1 (step=6)
Took action 1 got reward 1 (step=7)