MOLTE: a Modular Optimal Learning Testing Environment

Yingfei Wang
Princeton University

Warren Powell
Princeton University

Abstract

We address the relative paucity of empirical testing of learning algorithms (of any type) by introducing a new public-domain, Modular, Optimal Learning Testing Environment (**MOLTE**) for Bayesian ranking and selection problem, stochastic bandits or sequential experimental design problems. The Matlab-based simulator allows the comparison of a number of learning policies (represented as a series of .m modules) in the context of a wide range of problems (each represented in its own .m module) which makes it easy to add new algorithms and new test problems. State-of-the-art policies and various problem classes are provided in the package. The choice of problems and policies is guided through a spreadsheet-based interface. Different graphical metrics are included. **MOLTE** is designed to be compatible with parallel computing to scale up from local desktop to clusters and clouds. We offer **MOLTE** as an easy-to-use tool for the research community that will make it possible to perform much more comprehensive testing, spanning a broader selection of algorithms and test problems. We demonstrate the capabilities of **MOLTE** through a series of comparisons of policies on a starter library of test problems. We also address the problem of tuning and constructing priors that have been largely overlooked in optimal learning literature. We envision **MOLTE** as a modest spur to provide researchers an easy environment to study interesting questions involved in optimal learning.

Keywords: Bayesian optimization, ranking and selection, multi-armed bandits, sequential decision making, Matlab.

1. Introduction

We consider sequential decision problems in which at each time step, we choose one of finitely many alternatives and observe a random reward. The rewards are independent of each other and follow some unknown probability distribution. One goal can be to identify the alternative with the best expected performance within a limited measurement budget, which is the objective of offline ranking and selection. Another goal can be to maximize the expected cumulative sum of rewards obtained in a sequence of allocations, a problem class often addressed under the umbrella of multi-armed bandit problems.

Ranking and selection problems and/or multi-armed bandits arise in many settings. We may have to choose a type of material that has the best performance, the features in a laptop or car that produce the highest sales, or the molecular combination that produces the most effective drug. In health services, physicians have to make medical decisions (e.g. a course of drugs, surgery, and expensive tests) to provide the best treatment. In online advertisements, the system would like to choose ads to gather the most ad-clicks.

Since the seminal paper by Lai and Robbins (1985), there has been a long history in the optimal learning literature of proving some sort of bound, supported at times by relatively thin empirical work by comparing a few policies on a small number of randomly generated problems (Audibert, Bubeck *et al.* 2010; Cappé, Garivier, Maillard, Munos, Stoltz *et al.* 2013; Srinivas, Krause, Kakade, and Seeger 2009; Auer, Cesa-Bianchi, and Fischer 2002; Garivier and Moulines 2008; Audibert, Munos, and Szepesvári 2009). The problem, of course, is that compiling a library of test problems, and then running an extensive set of comparisons, is difficult. The problem is this means that we are analyzing the finite time performance of algorithms using bounds that only apply asymptotically by limited empirical experiments to support the claim of finite time performance. To this end, we introduce a new Modular Optimal Learning Testing Environment (**MOLTE**) for comparing a number of policies on a wide range of learning problems, providing the most comprehensive testbed that has yet appeared in the literature.

Similar libraries have been proposed for Bayesian optimization in different programming languages with different metrics and visualizations, for example, **BayesOpt** (Martinez-Cantin 2014) and **Spearmint** (Snoek, Larochelle, and Adams 2012). Yet the uniqueness of **MOLTE** lies in its design goal to facilitate comprehensive comparisons, on a broader set of test problems and a broader set of policies (which is not to restricted to Bayesian algorithms), rather than just a code library. With its unique modular design, **MOLTE** allows users to easily specify their own problems or their own algorithms without limitation as long as they follow the general function interface. The choice of problems and policies is guided through a spreadsheet-based interface. Since many of the algorithms have tunable parameters, we include the feature that the user can easily indicate in the spreadsheet to specify the value of the tunable parameter, or ask the package to optimize the tunable parameter. We have designed various (graphical) comparison metrics in order to gain a comprehensive understanding of different policies from different perspective. **MOLTE** is also designed to be compatible with parallel computing to scale up from local desktop to clusters and clouds. We offer **MOLTE** as an easy-to-use tool for the research community that provides a highly flexible environment for testing a range of learning policies on a library of test problems, so that researchers can more easily draw insights into the behavior of different policies in the context of different problem classes.

MOLTE is designed for problems where decisions can be represented as a set of discrete alternatives. These might be materials, drug combinations, features in a product, and medical decisions. They might also be discretized continuous decisions such as temperatures, pressures, concentrations, length and time (e.g. how long a material is soaked in a bath). If there are more than two or three dimensions, it is possible to use a sampled set of alternatives (this set can be updated from time to time, but **MOLTE** is designed to handle a single set of alternatives).

This paper is organized as follows. In Section 2, we lay out the mathematical models for sequential decision problems. Section 3 describes how **MOLTE** is implemented and describes the package from a user's perspective. We demonstrate the ability of **MOLTE** and the types of reports that it produces through extensive experimental results in Section 4 and 5. We draw the conclusion that there is no universal policy that works the best under all problem classes and the existence of bounds does not appear to provide reliable guidance regarding which policy works best. In practice, we believe that more useful guidance could be obtained by abstracting a real world problem, running simulations and using these to indicate which

policy works best. We envision **MOLTE** as a modest spur to induce other researchers to come forward to study interesting questions involved in optimal learning, for example, the issue of tuning as discussed in Section 6.

2. Sequential decision problems

Suppose we have a collection $\mathcal{X}$ of M alternatives, each of which can be measured sequentially to estimate its unknown performance $\mu_x = \mathbb{E}[F(x, W)]$. The utility function $F(x, W)$ can be understood as costs, rewards or losses, where $x \in \mathcal{X}$ is a decision variable and W is a random variable. The initial state S^0 is used to capture all information given as prior input. At each time step n, we use some policy to choose one alternative to measure $x^n = X^\pi(S^n)$ and receive a stochastic reward $\hat{F}^{n+1} = F(x^n, W^{n+1})$. After the decision and information, the system transitions to the state of belief at the next point in time according to some known transition function $S^{n+1} = S^M(S^n, x^n, \hat{F}^{n+1})$. Two styles of objective functions are considered in this paper:

- Terminal reward – considered in Bayesian optimization, ranking and selection problems, and also known as simple regret in multi-armed bandits. Here we assume have a limited budget of N function evaluations which have to be sequentially allocated over the different alternatives $x \in \mathcal{X}$ using a policy π. We use this policy to produce estimates $\theta_x^{\pi,N}$ of μ_x, and then choose the best design:

  $$X^{\pi,N} = \arg\max_x \theta_x^{\pi,N}.$$

  We can state the problem of finding the best experimental policy as

  $$\max_\pi \mathbb{E}\Big[F(X^{\pi,N}, W)|S^0\Big]. \tag{1}$$

  In this case, we are not punished for errors incurred during training and instead are only concerned with the final recommendation after the offline training phases. It should be noted that the expectation is over different sets of random variables. The first is the sequence of observations $W^1, \ldots, W^N$ which then produces the random $X^{\pi,N}$. The second expectation is over W in the equation, which is used to evaluate the solution. If a Bayesian approach is used, there is a third level of expectation over the prior.

- Cumulative reward – extensively studied under the umbrella of multi-armed bandits. If we have to experience the rewards while we do our learning/exploring, we may want to maximize contributions over some time horizon. The (online) objective function would be written as

  $$\max_\pi \mathbb{E}\Big[\sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1})|S^0\Big], \tag{2}$$

  where the expectation is over the sequence of observations $W^1, \ldots, W^N$ and the prior if any.

Despite different styles of objective functions, a general algorithm for sequential decision problems can be summarized in Algorithm 1:

Algorithm 1: General algorithm for sequential decision problems

input : time horizon N, initial state S^0, policy π, transition function S^M
for $n = 0$ *to* N **do**
  Select the point $x^n = X^\pi(S^n)$
  Observe $\hat{F}^{n+1} = F(x^n, W^{n+1})$
  Update the state $S^n = S^M(S^n, x^n, \hat{F}^{n+1})$
end

It should be noted that the states S^n can be understood as belief states which specifies the posterior on the unknown function $F(x, \cdot)$ if a Bayesian approach is used as in Bayesian optimization and ranking and selection problems. The transition function S^M then follows from Bayes' Theorem. We assume a normally distributed prior with heteroscedastic measurement noise (that is, the variance of an experiment depends on x), with known standard deviation σ_x^W. We begin with a normally distributed Bayesian prior belief $\mu_x \sim \mathcal{N}(\theta^0, \Sigma^0)$. For convenience, we introduce the σ-algebras $\mathcal{F}^n$ for any $n = 0, 1, ..., N-1$ which is formed by the previous n measurement choices and outcomes, $x^0, W^1, ..., x^{n-1}, W^n$. We define $\theta_x^n = \mathbb{E}[\mu_x|\mathcal{F}^n]$ and Σ^n the conditional covariance matrix. After a measurement W^{n+1} of alternative x, a posterior distribution on the beliefs are calculated by:

$$\theta^{n+1} = \Sigma^{n+1}\left((\Sigma^n)^{-1}\theta^n + \beta^W W^{n+1} e_x\right), \tag{3}$$

$$\Sigma^{n+1} = \left((\Sigma^n)^{-1} + \beta^W e_x e_x^T\right)^{-1}, \tag{4}$$

where e_x is the vector with 1 in the entry corresponding to alternative x and 0 elsewhere. $S^n = (\theta^n, \Sigma^n)$ is then our state of knowledge in this case.

3. Software implementation

In this section, we describe the implementation of **MOLTE**[1] that is designed to test a variety of different learning policies on a library of test problems. The architecture makes it particularly easy for researchers to add new policies, and new problems.

3.1. Structural overview

MOLTE is a Matlab-based modular architecture, where policies and problems are captured in a set of .m files, which makes it easy for researchers to add new policies and new problems. `MOLTE.m` compares the polices specified in an Excel spreadsheet for each problem class for `numP` times. Each time the simulator is run, it generates `numTruth` different sample paths, shared between all the policies, computes the value of the objective function for each sample path and then averages the `numTruth` replicas as the expected *terminal reward* or the expected *cumulative rewards*. The user may select in the spreadsheet to evaluate policies using either an online (cumulative reward) objective function Eq. (2), or an offline (terminal reward) objective function Eq. (1) (ranking and selection, Bayesian optimization).

In order to speed up the comparison, **MOLTE** is specially designed to be compatible with parallel computing to scale up from local desktop to clusters and clouds. This can be achieved

[1]The software is available at `http://www.castlelab.princeton.edu/software.htm`.

by first invoking `matlabpool` to submit a batch job to start a parallel environment and then use `parfor i=1:numP` instead of `for i=1:numP` on Line 111.

We pre-coded a number of standard test functions, including

- problems with additive Gaussian noise, such as
  - Branin's function (Dixon and Szegö 1978),
  - Goldstein Price function (Hu, Fu, and Marcus 2008),
  - Rosenbrock function (Hu *et al.* 2008),
  - Griewank function (Hu *et al.* 2008),
  - six-hump camel back function by (Molga and Smutnicki 2005)),
- synthetic bandit experiments (Audibert *et al.* 2010),
- Gaussian process regression,
- real-world applications like newsvendor problems and payload delivery.

We also pre-coded a number of competing policies, such as

- UCB variants (Auer *et al.* 2002),
- successive rejects,
- sequential Kriging (SKO, as a representative of Bayesian global optimization (Jones, Schonlau, and Welch 1998; Huang, Allen, Notz, and Zeng 2006; Jones 2001)),
- Thompson sampling,
- the knowledge gradient policies (Frazier, Powell, and Dayanik 2008).

Each of the problem classes and policies is organized in its own Matlab file, so that it is easy for a user to add in a new problem or a policy. In order to make a fair comparison, all the observations are pre-generated and shared between competing policies. There may be problems where a domain expert can provide prior knowledge (such as the likely success of a drug for a particular patient, the popularity of an online movie, or the sales potential of a set of features in a laptop), but in some settings these have to be derived from data. In **MOLTE** we provide various ways to construct a prior, including

- user-provided prior distributions,
- hard-coded default prior distributions,
- an uninformative prior,
- maximum likelihood estimation MLE (see Section 3.6).

3.2. Input Arguments

The input to the simulator is an Excel spreadsheet `ProblemsandAlgorithms.xls` which allows users to specify the problem classes and competing policies, as well as the belief models, the objectives, the prior construction and the measurement budgets. We provide a sample input spreadsheet in Table 1. For policies that have tunable parameters, a star included in the parentheses after the policy will initiate an automatic brute force tuning procedure with the optimal value reported in `alpha.txt`. The logic anticipates that tunable parameters may be anywhere from 10^{-5} up to 10^{5}. Whereas the user can also specify the value to be used for the policy in the parentheses.

Table 1: Sample input spreadsheet.

Problem class	**Prior**	**Measurement Budget**	**Belief Model**	**Offline/ Online**	**Number of policies**				
Bubeck1	Uninform	10	independent	Online	3	OLKG	IE(*)	UCB	
Branin	MLE	5	independent	Offline	4	UCBE(*)	IE(1.7)	KG	SR
GPR	Default	0.3	correlated	Online	4	KLUCB	EXPL	UCB	TS
NanoDesign	MLE	0.5	correlated	Offline	3	Kriging	EXPT	KG	

Problem class is the name of a pre-coded problem with a specified truth function, the number of alternatives and a default noise level. If it is a user defined problem, the user should write a .m file in the `./problemClasses` folder with the same name as presented in this spreadsheet. Due to the high popularity of Gaussian Process Regression (GPR), we offer the flexibility of directly specify the values of the parameter of GPR in the spreadsheet. For example, GPR$(\sigma, \beta; M)$ specifies the value of the parameters as follows (Powell and Ryzhov 2012): the prior mean θ_x^0 is drawn from $\mathcal{N}(0, \sqrt{\sigma})$, the prior covariance matrix Σ^0 is of the form $\sigma \exp(-\beta(x - x'))$ and M is the number of alternatives.

Prior indicates the ways to get a prior.

- *MLE* means using Latin hypercube designs and MLE for initial fit.
- *Default* can be used only for the problems (e.g. GPR and InanoparticleDesign) that have a default prior.
- *Given* means using the prior distribution provided by the user. It can be achieve either by specifying the parameters of the problem class, e.g. GPR(50, 0.45;100), or by providing a `Prior_problemClass.mat` file containing `mu_0`, `covM` and `beta_W` in the `./Prior` folder, e.g. `Prior_GPR.mat`.
- *Uninformative* specifies mean zero and infinite variance for each alternative.

Measurement Budget specifies the ratio between the time horizon of the decision making procedure to the number of alternatives. For example, in the spreadsheet a 5 means that the horizon will be 5 times the number of alternatives (which is 100), producing a total experimental budget of 500.

Belief Model specifies whether we are using independent or correlated beliefs for the policies which use a Bayesian belief model.

Offline/Online controls whether the objective is to maximize the expected terminal reward Eq. (1) or the expected cumulative rewards Eq. (2).

Number of Policies is the number of policies under comparison. This specifies the number of columns which contain the name of a policy to be tested, each represented in the corresponding `.m` file with the same name. If there are parentheses with a number after the name of the policy, it means setting the tunable parameter to the value specified in the parentheses. If there are parentheses with $*$, it means tuning the parameters and using the tuned value in the comparison; otherwise use the default value (in fact some policies, e.g. KG and Kriging, do not have tunable parameters). All policies are compared against the first policy in the list.

3.3. Output

All the data and figures are saved in a separate folder for each problem class. Within the folder of each problem class, each one of the `numP` folders (with the folder name from 1 to `numP`) contains:

`objectiveFunction.mat` saves the value of the online or offline objective function achieved by each policy for each of the `numP` replica.

`choice.mat` saves the decisions made by each policy in a variable named `choices` and the name of all policies in another variable `policies`. This file is only obtained for the first trial so that the dimension of `choices` is $number\_of\_policies \times M \times$ `numTruth`.

`FinalFit.mat` saves the final estimate of the surface by each policy after the measurement budget exhausted, together with the corresponding truth. This file is only obtained for the first trial.

`alpha.txt` saves the value of tunable parameter for each policy that requires tuning, i.e. with a $(*)$ in the input spreadsheet.

`offline_hist.pdf` is the histogram for each policy describing the distribution over `numP` trials of the expected terminal reward compared to the reward obtained by the reference policy (which is the first policy in the input spreadsheet).

`online_hist.pdf` is the histogram describing the distribution of the expected cumulative reward over `numP` trials. One of the example figure is Fig. 1. A distribution centered around a positive value implies the policy underperforms the reference policy, which in this example is UCBV.

It is always useful for researchers to examine the sampling pattern of each policy to gain a better understanding of its behavior. To this end, we provide a function `histChoice.m` that reads in the `choice.mat` and generates the distribution of the frequency of choosing each of the alternatives for each policy. `filedir` specifies which one of the `numP` trials is used to generate the sampling pattern, e.g. `filedir='./1/'`. Since within each trial, `numTruth` different truths are sampled, `numT` is used to indicate the number of truths the user would like to draw the sampling pattern from. Fig. 2 is an example of a sampling pattern with the x-axis the 100 alternatives and the histogram of the sampling pattern under a measurement budget of 300.

We also provide other graphical metrics for comparing the policies. `genProb.m` can read in the `objectiveFunction.mat` and depict the mean opportunity cost with error bars indicating the standard deviation of each policy as shown in Fig. 3(a), together with the probability of each policy being optimal and being the best in Fig. 3(b):

The statistics stored in `objectiveFunction.mat`, `choice.mat` and `FinalFit.mat` can easily be used for other illustrations. For example, one can use the truth values stored in

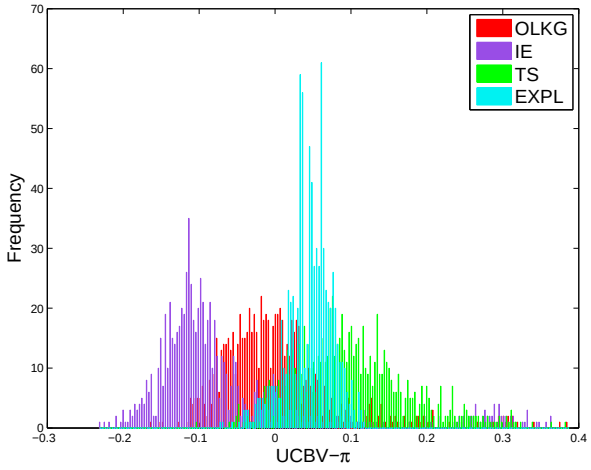


Figure 1: Example figure of online_hist.pdf.

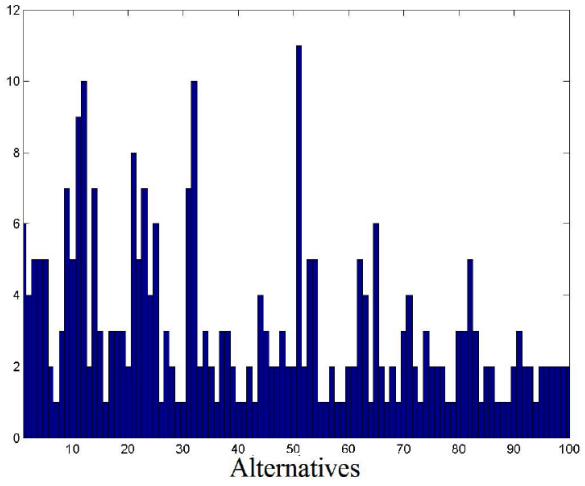


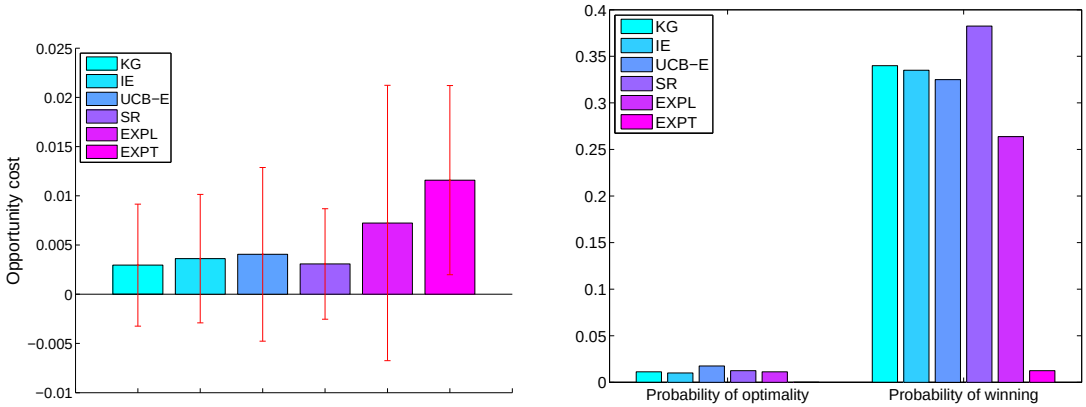
Figure 2: Example figure of the histogram of the frequency of choosing each of the alternative under a policy.

`FinalFit.mat` and the number of times each policy samples each alternative in `choice.mat` to generate two dimensional contour plot using Matlab commands `contour(...)`, `plot(...)` and `text(...)`, as well as the corresponding posterior contour using the final estimate of the surface stored in `FinalFit.mat`, as we demonstrate later in Fig. 5.

3.4. Pre-coded problem classes

While a wide range of problem classes and policies are precoded in **MOLTE**, in the next two subsections we only briefly summarize the problem classes and policies mentioned in the following numerical experiments of this paper. As of this writing, **MOLTE** includes 23 pre-coded problem classes, and 20 pre-coded policies.

Bubeck's Experiments: (Audibert *et al.* 2010) We consider Bernoulli distributions with the mean of the best arm always $\mu_1 = 0.5$. M is the number of arms.



(a) Opportunity cost (b) Probability of optimality/winning

Figure 3: (a) depicts the mean opportunity cost with error bars indicating the standard deviation of each policy. The first bar group in (b) demonstrates the probability that the final recommendation of each policy is the optimal one. The second bar group in (b) illustrates the probability that the opportunity cost of each policy is the lowest.

Bubeck1: $M = 20$, $\mu_{2:20} = 0.4$.
Bubeck2: $M = 20$, $\mu_{2:6} = 0.42$, $\mu_{7:20} = 0.38$.
Bubeck3: $M = 4$, $\mu_i = 0.5 - (0.37)^i$, $i \in \{2, 3, 4\}$.
Bubeck4: $M = 6$, $\mu_2 = 0.42$, $\mu_{3:4} = 0.4$, $\mu_{5:6} = 0.35$.
Bubeck5: $M = 15$, $\mu_i = 0.5 - 0.025i, i \in \{2, \cdots, 15\}$.
Bubeck6: $M = 20$, $\mu_2 = 0.48$, $\mu_{3:20} = 0.37$.
Bubeck7: $M = 30$, $\mu_{2:6} = 0.45$, $\mu_{7:20} = 0.43$, $\mu_{21:30} = 0.38$.

Asymmetric unimodular function (AUF): x is a controllable parameter ranging from 21 to 120. The objective function is $F(x, \xi) = \theta_1 \min(x, \xi) - \theta_2 x$, where θ_1, θ_2 and the distribution of the random variable ξ are all unknown. ξ is taken as a normal distribution with mean 60. Three noise levels are considered by setting different noise ratios between the standard deviation and the mean of ξ: HNoise–0.5, MNoise–0.4, LNoise–0.3. Unless explicitly pointed out, experiments are taken under LNoise.

Equal-prior: $M = 100$. The true values μ_x are uniformly distributed over $[0, 60]$ and measurement noise $\sigma_W = 100$. $\theta_x^0 = 30$ and $\sigma_x^0 = 10$ for every x.

All the standard optimization test functions are flipped in MOLTE to generate maximization problems instead of minimization in line with R&S and bandit problems. The standard deviation of the additive Guassion noise is set to 20 percent of the range of the function values.

Rosenbrock functions with additive noise:

$$f(x, y, \phi) = 100(y - x^2)^2 + (1 - x)^2 + \phi,$$

where $-3 \le x \le 3$, $-3 \le y \le 3$. x and y are uniformly discretized into 13 $\times$ 13 alternatives.

Pinter's function with additive noise:

$$f(x, y, \phi) = \log_{10}\left(1 + (y^2 - 2x + 3y - \cos x + 1)^2\right) + \log_{10}\left(1 + 2(x^2 - 2y + 3x - \cos y + 1)^2\right) + x^2 + 2y^2 + 20\sin^2(y \sin x - x + \sin y) + 40\sin^2(x \sin y - y + \sin x) + 1 + \phi,$$

where $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. x and y are uniformly discretized into 13×13 alternatives.

Goldstein-Price's function with additive noise:

$$\begin{aligned} f(x,y,\phi) &= [1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)] \cdot \\ &\quad [30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)] + \phi, \end{aligned}$$

where $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. x and y are uniformly discretized into 13×13 alternatives.

Branins's function with additive noise:

$$f(x,y,\phi) = (y - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6)^2 + 10(1-\frac{1}{8\pi})\cos(x) + 10 + \phi,$$

where $-5 \leq x \leq 10$, $0 \leq y \leq 15$. x and y are uniformly discretized into 15×15 alternatives.

Ackley's function with additive noise:

$$f(x,y,\phi) = -20\exp\Big(-0.2\cdot\sqrt{\frac{1}{2}(x^2+y^2)}\Big) - \exp\Big(\frac{1}{2}(\cos(2\pi x)+\cos(2\pi y))\Big) + 20 + \exp(1) + \phi,$$

where $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. x and y are uniformly discretized into 13×13 alternatives.

Hyper Ellipsoid function with additive noise:

$$f(x,y,\phi) = x^2 + 2y^2 + \phi.$$

where $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. x and y are uniformly discretized into 13×13 alternatives.

Rastrigin function with additive noise:

$$f(x,y,\phi) = 20 + \big[x^2 - 10\cos(2\pi x)\big] + \big[x^2 - 10\cos(2\pi y)\big] + \phi,$$

where $-3 \leq x \leq 3$, $-3 \leq y \leq 3$. x and y are uniformly discretized into 11×11 alternatives.

Six-hump camel back function with additive noise:

$$f(x,y,\phi) = (4 - 2.1x^2 + \frac{x^4}{3})x^2 + xy + (-4+4y^2)y^2 + \phi,$$

where $-2 \leq x \leq 2$, $-1 \leq y \leq 1$. x and y are uniformly discretized into 13×13 alternatives.

3.5. Pre-coded policies

We have pre-coded various state-of-the-art policies π, which differ according to their decision $X^{\pi,n}(S^n)$ of the alternative to measure at time n given state S^n.

Knowledge gradient (KG): (Frazier *et al.* 2008; Frazier, Powell, and Dayanik 2009) This policy is designed for offline objective (1). Define the knowledge gradient as

$$\nu_x^{\mathrm{KG},n} = \mathbb{E}[\max_{x'} \theta_{x'}^{n+1} - \max_{x'} \theta_{x'}^n | x^n = x, S^n].$$

$$X^{\mathrm{KG},n}(S^n) = \arg\max_{x\in\mathcal{X}} \nu_x^{\mathrm{KG},n}.$$

Online knowledge gradient (OLKG): (Ryzhov, Powell, and Frazier 2012)

$$X^{\mathrm{OLKG},n}(S^n) = \arg\max_{x\in\mathcal{X}} \theta_x^n + (N-n)\nu_x^{\mathrm{KG},n}.$$

Interval Estimation (IE): (Kaelbling 1993)

$$X^{\text{IE},n}(S^n) = \arg\max_x \theta_x^n + z_{\alpha/2}\sigma_x^n,$$

where $z_{\alpha/2}$ is a tunable parameter.

Kriging: Huang *et al.* (2006)

Let $x^* = \arg\max_x(\theta_x^n + \sigma_x^n)$, and then

$$X^{\text{Kriging},n}(S^n) = \arg\max_x(\theta_x^n - \theta_{x^*}^n)\Phi(\frac{\theta_x^n - \theta_{x^*}^n}{\sigma_x^n}) + \sigma_x^n\phi(\frac{\theta_x^n - \theta_{x^*}^n}{\sigma_x^n}),$$

where ϕ and Φ are the standard normal density and cumulative distribution functions.

Thompson sampling (TS): (Thompson 1933)

$$X^{\text{TS},n}(S^n) = \arg\max_x \tilde{\theta}_x^n,$$

where $\tilde{\theta}_x^n \sim \mathcal{N}(\theta_x^n, \sigma_x^n)$ for independent beliefs or $\tilde{\theta}_x^n \sim \mathcal{N}(\theta^n, \Sigma^n)$ for correlated beliefs.

UCB: (Auer *et al.* 2002)

$$X^{\text{UCB},n}(S^n) = \arg\max_x \hat{\theta}_x^n + \sqrt{\frac{2V_x^n \log n}{N_x^n}},$$

where $\hat{\theta}_x^n$, V_x^n, N_x^n are the sample mean of μ_x, sample variance of μ_x, and number of times x has been sampled up to time n, respectively. The quantity $\hat{\theta}_x^0$ is initialized by measuring each alternative once. These are similarly defined in the following variants of UCB.

UCB-E: (Audibert *et al.* 2010)

$$X^{\text{UCB-E},n}(S^n) = \arg\max_x \hat{\theta}_x^n + \sqrt{\frac{\alpha}{N_x^n}},$$

where α is a tunable parameter.

UCB-V: (Audibert *et al.* 2009)

$$X^{\text{UCB-V},n}(S^n) = \arg\max_x \hat{\theta}_x^n + \sqrt{\frac{V_x^n \log n}{N_x^n}} + 1.5\frac{\log n}{N_x^n}.$$

SR: (Audibert *et al.* 2010) Let $A_1 = \mathcal{X}$, $\overline{\log}(M) = \frac{1}{2} + \sum_{i=2}^{M} \frac{1}{i}$,

$$n_m = \left\lceil \frac{1}{\overline{\log}(M)} \frac{n - M}{M + 1 - m} \right\rceil.$$

For each phase $m = 1, ..., M-1$:

1. For each $x \in A_m$, select alternative x for $n_m - n_{m-1}$ rounds.
2. Let $A_{m+1} = A_m \setminus \arg\min_{x \in A_m} \hat{\theta}_x$.

KLUCB: (Cappé *et al.* 2013)

$$X^{\mathrm{KLUCB},n}(S^n) = \arg\max_x \hat{\theta}_x^n + \sqrt{\frac{2V_x^n(\log n + 3\log\log(n))}{N_x^n}}.$$

EXPL: A pure exploration strategy that tests each alternative equally often through random sampling of the set of alternatives.

EXPT: A pure exploitation strategy.

$$X^{\mathrm{EXPT},n}(S^n) = \arg\max_x \hat{\theta}_x^n.$$

3.6. Prior Generation

MOLTE features the following strategies for building a prior:

- If an *uninformative* prior is specified by the user for independent beliefs, a uniform prior will be used with $\theta_x^0 = 0$ and $\sigma_x^0 = inf$ for every x. In such case, same as with frequentist approaches (for example, UCBs), Bayesian approaches will measure each alternative once at the very beginning.
- User-defined priors can be achieved either by specifying the parameters of the problem class, e.g. GPR(50, 0.45;100), or by providing a `Prior_problemClass.mat` file containing `mu_0`, `covM` and `beta_W` in the `./Prior` folder, e.g. `Prior_GPR.mat`.
- If maximum likelihood estimation (*MLE*) is chosen to obtain the prior distribution for either independent beliefs or correlated beliefs, we follow Jones *et al.* (1998) and Huang *et al.* (2006) to use Latin hypercube designs for initial fit. For independent beliefs, we adopt a uniform prior with the same mean value θ_x^0 and standard deviation σ_x^0 for all alternatives. For correlated beliefs, we use a constant mean value θ_x^0 for all alternatives and a prior covariance matrix of the form

  $$\Sigma_{xx'}^0 = \sigma e^{-\sum_{i=1}^d \lambda_i (x_i - x_i')^2},$$

  where each arm x is a d-dimensional vector and σ, λ_i are constant. We adopt the rule of thumb by Jones *et al.* (1998) for the default number $(10 \times p)$ of points, where p is the number of parameters to be estimated. In addition, as suggested by Huang *et al.* (2006), to estimate the random errors, after the first $10 \times p$ points are evaluated, we add one replicate at each of the locations where the best p responses are found. Maximum likelihood estimation is then used to estimate the parameters based on the points in the initial design.

3.7. User defined problem classes and/or policies

Each of the problem classes is organized in its own .m file in the `./problemClasses` folder. The standard API is defined as:

```
function [mu, beta_W, numD]=UserDefinedName(varargin)
%user defined function
```

```
end
```

where varargin is used to pass input parameters with variable lengths for the problem class if needed. `mu` is a column vector generating a true function value (not known to the learner) of the user defined problem class. If Gaussian process regression is used, `mu` is sampled from the prior distribution. `beta_W` is a column vector of the inverse of the variance of measurement noise $\sigma_x^{W,2}$ and `numD` specifies the dimensionality of the function. For example, if the user is intended to define their own GPR problems, the function should be defined as:

```
function [mu, beta_W, numD] = UserDefinedName( varargin )}
     mu_0=varargin{1};
     covM=varargin{2};
     beta_W=varargin{3};
     mu=mvnrnd(mu_0, covM)';
     numD=1;
end
```

The user also needs to provide the priors for the GPR problems. This can be done by writing a `Prior_UserDefinedName.m` file in the `./Prior` folder to provide the prior mean, covariance matrix and the measurement noise:

```
function [mu_0, covM, beta_W] = Prior_UserDefinedName( varargin )
```

If the user would like to define a problem from a test function (such as Pinter's function with additive noise introduced in Section 3.4), one should first code up the test function, choose a discrete set of alternatives and calculate their function values. One example can be:

```
function [mu, beta_W, numD] = Pinter( varargin )
     %set the dimensionality of the function
     numD=2;

     %choose the discrete set of alternatives of interest
     xx=-3:0.5:3;
     yy=-3:0.5:3;
     [x,y]=meshgrid(xx,yy);
     f=fun(x,y);
     [a,b]=size(f);
     mu=max(max(f))-reshape(f,[a*b,1]);

     %define the variance of measurement noise
     beta_W=1./(0.2*(max(max(f))-min(min(f)))).^2*ones(length(mu),1);
end

%define the test function
function f=fun(x,y)
      f=x.^2+2*y.^2+20*sin(y.*sin(x)-x+sin(y)).^2+40*sin(x.*sin(y)-y+sin(x)).^2
       +log10(1+(y.^2-2*x+3*y-cos(x)+1).^2)
       +2*log10(1+2*(x.^2-2*y+3*x-cos(y)+1).^2)+1+rand()*5;
end
```

Each of the policies is also organized in its own .m file in the ./policies folder. The standard API is defined as:

```
function [mu_est, count, recommArm] = Name( mu_0,beta_W,covM,samples,alpha,tune)
     [M,N]=size(samples);
     count=zeros(M,1);  % count the times each alternative is measured
     mu_est=mu_0;
     for i = 1:N
          %user defined policy decision rule to find x=X^{\pi}(S^n)
          count(x)=count(x)+1;
          W=samples(x, count(x));
          %user defined transition function S^{n+1} = S^{M}(S^n, x,W)
     end
     [value, recommendedArm] = max(mu_est);
end
```

where `mu_0` and `covM` specifies the prior distribution, `beta_W` is the known measurement (experimental) noise, `samples` are pre-generated and shared among all the policies, `alpha` is the tunable parameter and `tune` specifies whether to tune this policy or use default value.

4. Experiments for Offline (Terminal Reward) Problems

In this section we report on a series of experiments with the goal of illustrating the use of **MOLTE** and the types of reports that it produces. We do not attempt to demonstrate that any policy is better than another, but our experiments support the hypothesis that different policies work well on different problem classes. This observation supports the claim that more careful empirical work is needed to develop a better understanding of which policies work best, and under what conditions.

4.1. Experiments with independent beliefs

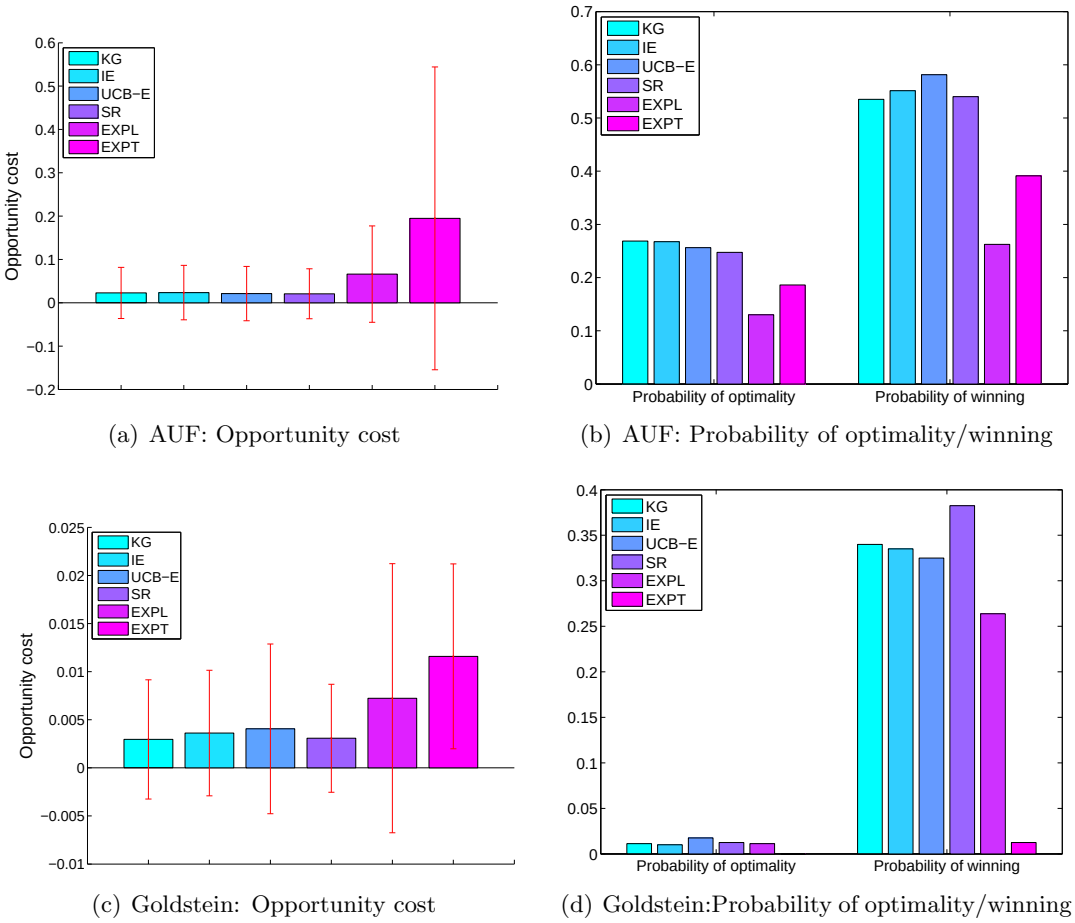
We first compare the performance of KG, IE with tuning, UCB-E with tuning, SR, EXPL and EXPT for offline ranking and selection problems. MLE is used to construct the prior distribution for KG and IE. Figure 4 shows the performance in problem classes AUF and Goldstein with independent beliefs under a measurement budget five times the number of alternatives.

We run each policy for `numP=1000` times. We illustrate in the first column of Figure 4 the mean opportunity cost and the standard deviation of each policy over 1000 runs, with the opportunity cost (OC^{π}) defined as:

$$\mathrm{OC}^{\pi} = \max_{x} \mu_x - \mu_{x^{\pi,N}},$$

where $x^{\pi,N} = \arg\max_x \theta_x^{\pi,N}$.

In order to provide a more comprehensive comparison of different policies, we also calculate the probability that the final recommendation of each policy is the optimal one and the probability that the opportunity cost of each policy is the lowest, as illustrated in the figures on the right hand side of Figure 4.



(a) AUF: Opportunity cost

(b) AUF: Probability of optimality/winning

(c) Goldstein: Opportunity cost

(d) Goldstein:Probability of optimality/winning

Figure 4: Comparisons for AUF and Goldstein. (a) and (c) depict the mean opportunity cost with error bars indicating the standard deviation of each policy. The first bar group in (b) and (d) demonstrates the probability that the final recommendation of each policy is the optimal one. The second bar group in (b) and (d) illustrates the probability that the opportunity cost of each policy is the lowest.

The three criteria characterize the behavior of policies in different aspects. For example, under AUF, if one cares about the average performance of the policy and its stability, SR is the best choice concluding from Figure 4 (a). Yet, if one can only run one trial (as in most cases of experimental science) and want to identify the best alternative, KG might be a better choice since it has the highest probability of finding the optimal alternative. Or if one can live with fairly good alternatives other than the optimal one, UCB-E could be the choice (although it has to be carefully tuned).

One observation is that there is no universal best policy for all problem classes or under all criteria. In practice, a useful guidance could be abstracting the real world problem and running synthetic simulations to find the best simulated policy under some desired criterion before conducting the real experiments.

4.2. Experiments with correlated beliefs

In this section, we exploit correlated beliefs between alternatives in order to strengthen the effect of each measurement so that one measurement of some alternative can provide information for other alternatives.

In order to better understand the behavior of each policy, a useful way is to examine the sampling pattern of each policy. We present an example of the frequency of measuring each alternative for each competing policy for Branin functions with a measurement budget of 100. To take advantage of correlated beliefs, rather than measuring each alternative once to initialize the empirical mean, we use the prior mean as the starting point and use the posterior mean θ^n in place of the empirical mean $\hat{\theta}^n$ for UCB-E. In the left column of Figure 5, the sampling pattern of each policy is displayed together with the contour of the Branin objective function which exhibits one global maximum at $(-3, 12)$ and other two local maxima at $(9, 3)$ and $(16, 4)$. The frequency that each alternative is measured is marked in numbers. The right column depicts the final prediction under each policy. All the observations are pre-generated and shared for all policies. We see from the figures that since KGCB and Kriging take correlation into consideration in the decision functions, they need less exploration and rely on the correlation to provide information for less explored alternatives. They quickly begin to focus on the alternatives that have the best values. Yet Kriging wanders around local minima for a while before it heads toward the global maximum. Note that the prediction of KGCB gives a good match in general. The function value at the true maximum alternative is well approximated, while moderate error in the estimate is located away from this region of interest. UCB-E is exploring more than necessary and wasting time on less promising regions. But when the budget is big enough, the exploration will contribute to better prediction of the surface, leading to a potentially larger final outcome in the long run. Pure exploitation gets stuck in a seemingly good alternative and the sampling pattern is not reasonable nor meaningful.

5. Experiments for Online (Cumulative Reward) Problems

In this section, we provide sample comparisons of different policies using the online (cumulative reward) objective function. The performance measure that we use to evaluate a policy π

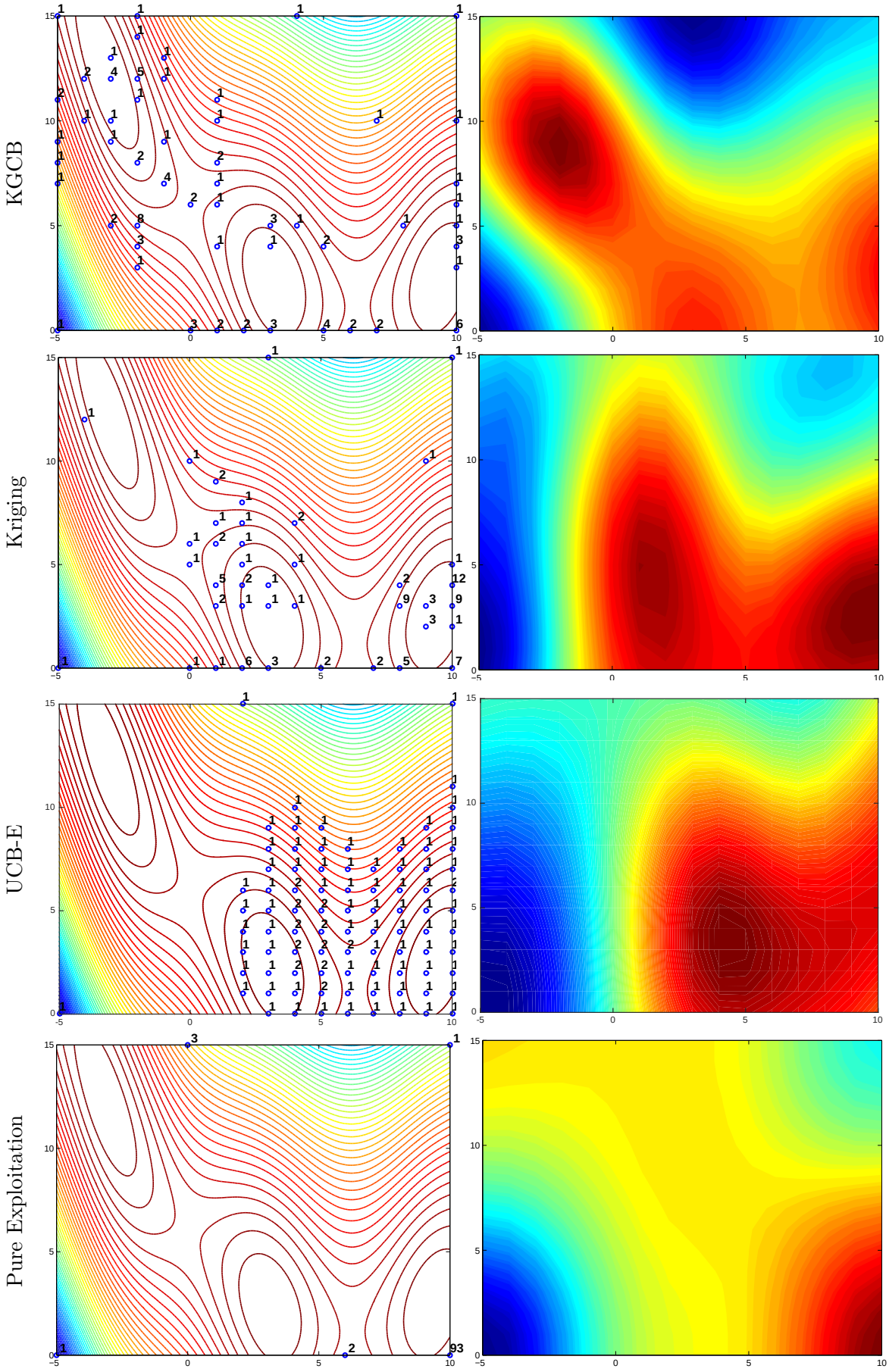


Figure 5: Left column: sampling distribution. Right column: posterior distribution.

in an online setting is $\frac{\bar{R}_N^\pi}{N}$, where the pseudo-regret $\bar{R}_N^\pi$ is defined as

$$\bar{R}_N^\pi = N \max_{x \in \mathcal{X}} \mu_x - \sum_{n=0}^{N} \mathbb{E}[\mu_{X^{\pi,n}(s^n)}].$$

The opportunity cost (OC) between two policies in an online setting is defined as the difference of their pseudo-regrets.

5.1. Experiments with independent beliefs

In real world problems, especially in experimental science, frequentist techniques cannot incorporate prior knowledge from domain experts, relying instead on the training from vast pools of data. This may be infeasible to perform in reality since running one experiment might be very expensive. The advantage of a Bayesian approach is unarguable in such cases. However, if we use MLE to fit the prior instead of using domain knowledge, it seems that the comparisons are in favor of Bayesian approaches by using an extra $11 \times p$ measurements. In order to make a seemingly more fair comparison in our synthetic experimental setting, we also experiment with uninformative priors with no additional information provided for Bayesian approach.

Tables 2, 3 and 4 provide comparisons of OLKG, IE with tuning, UCB-E with tuning, UCB, UCB-V, KLUCB, pure exploration (EXPL) using the Bubeck problems with uninformative prior. The measurement budgets are set to 10, 100 and 500 times the number of alternatives of each problem class in Tables 2, 3 and 4, respectively. IE and UCB-E are carefully tuned for each problem class. Under each problem class, we ran each policy for `numP=1000` times. In each run, all the measurements are pre-generated and shared across all the policies. For each policy we record the normalized opportunity cost between OLKG and other competing policies, where the normalized opportunity cost is defined as the ratio between the opportunity cost $\frac{\bar{R}_N^\pi}{N} - \frac{\bar{R}_N^{\text{OLKG}}}{N}$ and the range of the truth μ. Positive values of OC indicate that the corresponding policy underperforms OLKG on average. Other than the interest of average performance measured by pseudo-regret, only one sample path will be realized in real world experiments and it is meaningful to find out which policy is most likely to perform the best in one sample run. Thus we also report the probability that each of the other policy outperforms (obtains a lower regret than) OLKG within 1000 realizations. Any policy can be set as a benchmark by placing it as the first policy in the input spreadsheet.

We see from the three tables that the probability of any other policy that outperforms OLKG is in general much less than 0.5. If this criterion is what an experimenter anticipates, then OLKG is a safe choice in most situations. We then discuss the performance of each policy in terms of OC. At the beginning of each trial, IE and UCB-E are more exploiting than exploring while OLKG tends to explore before it moves toward the best estimates. This contributes to good performance (measured by OC) of IE and UCB-E in Table 2 with a small measurement budget. The tuned values of parameters further sharpen this effect by utilizing smaller values compared to those under larger measurement budgets as reported in Table 5 which summarizes the optimally tuned values for each parameter. Since UCB policies tend to explore more than necessary (which can be seen from the sampling pattern, for example, Figure 5), the performance degenerates with a moderate measurement budget as shown in Table 3. In this case, OLKG yields the best performance since after an exploration

Table 2: The difference between each policy and OLKG (OC), and the probability that each policy outperforms OLKG, using uninformative priors with a measurement budget 10 times the number of alternatives.

Problem Class	IE		UCBE		UCBV		UCB		KLUCB		EXPL	
	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Bubeck1	-0.031	0.43	-0.032	0.43	0.073	0.51	0.016	0.35	0.054	0.50	0.078	0.50
Bubeck2	-0.032	0.55	-0.031	0.52	0.097	0.30	0.025	0.43	0.070	0.35	0.105	0.29
Bubeck3	-0.000	0.29	0.006	0.30	0.068	0.26	0.021	0.53	0.020	0.34	0.095	0.23
Bubeck4	-0.004	0.39	-0.003	0.57	0.100	0.36	0.029	0.48	0.040	0.40	0.124	0.33
Bubeck5	-0.019	0.71	-0.020	0.71	0.213	0.01	0.018	0.48	0.087	0.11	0.255	0.00
Bubeck6	-0.034	0.49	-0.035	0.48	0.139	0.34	0.034	0.41	0.098	0.37	0.151	0.33
Bubeck7	-0.036	0.70	-0.036	0.71	0.065	0.17	0.009	0.48	0.043	0.22	0.073	0.15

Table 3: The difference between each policy and OLKG (OC), and the probability that each policy outperforms OLKG, using uninformative priors with a measurement budget 100 times the number of alternatives.

Problem Class	IE		UCBE		UCBV		UCB		KLUCB		EXPL	
	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Bubeck1	0.006	0.34	0.015	0.32	0.387	0.36	0.245	0.14	0.311	0.37	0.431	0.36
Bubeck2	0.006	0.31	0.017	0.35	0.399	0.09	0.226	0.17	0.309	0.22	0.458	0.06
Bubeck3	0.002	0.32	0.007	0.31	0.111	0.18	0.077	0.39	0.052	0.25	0.214	0.07
Bubeck4	-0.014	0.31	-0.005	0.30	0.232	0.27	0.156	0.32	0.114	0.30	0.365	0.17
Bubeck5	-0.003	0.39	0.003	0.34	0.228	0.01	0.064	0.26	0.094	0.15	0.425	0.00
Bubeck6	0.014	0.38	0.025	0.38	0.522	0.10	0.274	0.12	0.380	0.10	0.619	0.09
Bubeck7	0.015	0.52	0.016	0.44	0.260	0.00	0.158	0.21	0.215	0.09	0.303	0.00

period, it begins to focus on the alternatives that have the best estimates while looking for alternatives whose estimates are less certain. Yet exploration benefits in the long run. Thus the performance of UCB policies and IE improves if allowed to explore for a sufficiently long time as reported in Table 4.

5.2. Experiments with correlated beliefs

In this section, we summarize numerical experiments on problems with correlated beliefs between different policies, including OLKG, IE with tuning, UCBE, UCB-V, Kriging, UCB, Thompson Sampling (TS) and pure exploration (EXPL). To take advantage of correlated beliefs, we use the prior mean as the starting point and use posterior mean θ^n in place of the empirical mean for UCB-V and UCB policies.

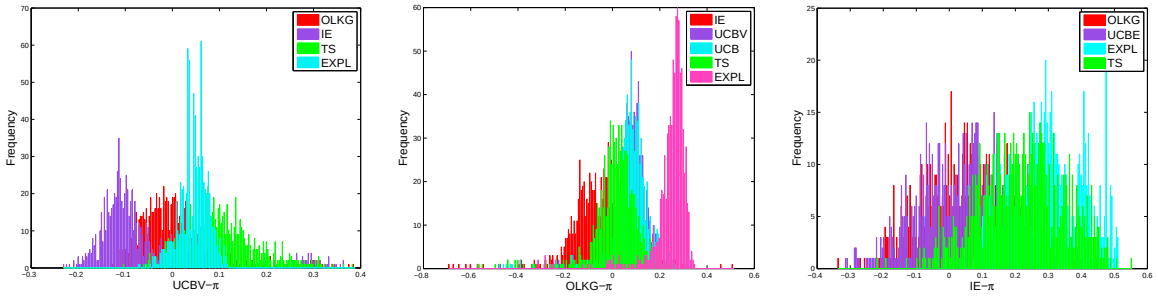
In order to gain a good understanding of the performance of the policies, MOLTE produces histograms illustrating the distribution of the difference between the normalized OC of a benchmark policy and either of the other policies over 1000 runs. Whichever policy that is listed as the first policy is treated as the benchmark. The measurement budget is set to 0.2 times the number of alternatives of each problem class. Figure 6 compares the performance of several policies under various problem classes with different benchmark policies. A distribution centered around a positive value implies the policy underperforms the benchmark policy,

Table 4: The difference between each policy and OLKG (OC), and the probability that each policy outperforms OLKG, using uninformative priors with a measurement budget 500 times the number of alternatives.

Problem Class	IE		UCBE		UCBV		UCB		KLUCB		EXPL	
	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Bubeck1	-0.105	0.30	-0.098	0.30	0.296	0.26	0.288	0.10	0.175	0.27	0.634	0.26
Bubeck2	-0.089	0.28	-0.080	0.26	0.253	0.31	0.226	0.15	0.139	0.32	0.609	0.02
Bubeck3	-0.009	0.34	-0.006	0.31	0.069	0.18	0.077	0.39	0.035	0.29	0.268	0.03
Bubeck4	-0.075	0.28	-0.069	0.27	0.091	0.26	0.174	0.24	0.014	0.26	0.462	0.12
Bubeck5	-0.030	0.33	-0.026	0.31	0.066	0.28	0.050	0.23	0.012	0.34	0.462	0.00
Bubeck6	-0.024	0.26	-0.022	0.24	0.310	0.05	0.227	0.16	0.190	0.06	0.771	0.05
Bubeck7	-0.045	0.33	-0.045	0.34	0.262	0.11	0.152	0.23	0.200	0.27	0.430	0.00

Table 5: Tuned parameters of IE and UCB-E under different problem classes and measurement budgets. The second row indicates the ratio between the measurement budget and the number of alternatives.

Problem Class	IE			UCBE		
	10	100	500	10	100	500
Bubeck1	0.0007079	1.295	2.036	0.0008991	0.3934	1.103
Bubeck2	0.1675	1.295	2.169	0.002359	0.337	0.9063
Bubeck3	0.8991	1.395	1.878	0.1206	0.4562	0.8635
Bubeck4	0.8991	1.571	2.196	0.004392	0.5332	1.197
Bubeck5	0.004566	1.395	2.169	0.0003102	0.3518	1.002
Bubeck6	0.09063	1.197	1.642	0.000505	0.3201	0.7748
Bubeck7	0.002773	0.8991	1.878	0.0005936	0.2169	0.8007



(a) Goldstein (b) HyperEllipsoid (c) Rastrigin

Figure 6: Normalized opportunity cost between different policies.

Table 6: Comparisons with OLKG for correlated beliefs with the measurement 0.2 times the number of alternatives of each problem class.

Problem Class	IE		UCBE		UCBV		Kriging		TS		EXPL	
	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Goldstein	-0.061	0.81	-0.097	0.92	-0.003	0.45	-0.031	0.73	0.100	0.09	0.041	0.16
AUF_HNoise	0.058	0.40	0.022	0.43	0.037	0.54	0.031	0.39	0.073	0.22	0.047	0.48
AUF_MNoise	0.043	0.29	0.027	0.42	0.343	0.21	0.023	0.28	0.173	0.21	-0.057	0.52
AUF_LNoise	-0.043	0.73	-0.013	0.64	0.053	0.51	0.005	0.53	0.038	0.20	0.003	0.62
Branin	-0.027	0.76	0.025	0.24	0.026	0.26	0.004	0.54	0.041	0.07	0.123	0.00
Ackley	0.007	0.42	0.04	0.41	0.106	0.20	0.037	0.42	0.100	0.23	0.344	0.00
HyperEllipsoid	-0.059	0.73	0.064	0.12	0.08	0.07	0.146	0.22	0.011	0.38	0.243	0.03
Pinter	-0.028	0.56	-0.003	0.51	0.029	0.42	-0.055	0.65	0.122	0.19	0.177	0.04
Rastrigin	-0.082	0.70	-0.03	0.56	0.162	0.04	-0.026	0.57	0.136	0.08	0.203	0.01

while one centered around a negative number means the policy outperforms the benchmark. For example, Figure 6(a) compares the performance of UCBV, OLKG, IE, TS and EXPL under Goldstein with UCBV as the benchmark policy. We can see that the tuned IE and OLKG are outperforming UCBV and others are underperforming.

We close this section by providing more comparisons between other policies with OLKG under various problem classes. The measurement budget is set to 0.2 times the number of alternatives of each problem class. Table 6 reports the normalized mean OCs and the probability that each of the other policy outperforms OLKG under 1000 runs. IE and UCB-E are carefully tuned for each problem classes with the optimal value shown in Table 7. IE and UCB-E after tuning works generally well. Yet the optimal values of the tuned parameters are quite different for different problems as shown in Table 5 and 7. In addition, the performance of the policies are sensitive to the value of the tunable parameters. In light of this issue, we can conclude that OLKG and Kriging have one attractive advantage over IE and UCB-E: they require no tuning at all, while yielding comparable performance to a finely tuned IE or UCB-E policy. A detailed study on the issue of tuning is presented in Section 6.1.

Table 6 together with the comparisons shown in previous sections suggests that there is no universal best policy for all problem classes and one could possibly design toy problems for either policy to perform the best. Similar observations have also been reported by Kuleshov

Table 7: Tuned parameters of IE and UCB-E under different problem classes.

Problem Class	IE	UCBE
Goldstein	0.009939	2571
AUF_HNoise	0.01497	0.319
AUF_MNoise	0.01871	1.591
AUF_LNoise	0.01095	6.835
Branin	0.2694	0.0003664
Ackley	1.197	1.329
HyperEllipsoid	0.8991	21.21
Pinter	0.9989	0.0001636
Rastrigin	0.2086	0.001476

and Precup (2000) for different bandit problems on different metrics. Besides, there are theoretical guarantees proved for each of the policy mentioned above, but the existence of these bounds does not appear to provide reliable guidance regarding which policy works best. An asymptotic bound does not provide any assurance that an algorithm will work well on a particular problem in finite time. In practice, we believe that more useful guidance could be obtained by abstracting a real world problem, running simulations and using these to indicate which policy works best.

6. Discussion

We close our presentation by discussing two issues that tend to be overlooked in comparisons of learning algorithms: the tuning of heuristic parameters (widely used in frequentist UCB policies) and priors (used in all Bayesian policies such as knowledge gradient).

6.1. The issue of tuning

Previous experimental results show that tuned version of IE and UCB-E yield good performance in general and yet the optimal value for IE and UCB-E may be highly problem dependent. Our experiments also suggest that the performance of a policy is sensitive to the value of the tuned parameter. For example, Figure 8 provides the comparisons between the performances of IE with different parameter values (provided in the parentheses) with the online objective function under various problem classes. The measurement budget is set to five times the number of alternatives for each problem class experimented with independent beliefs and 0.3 times the number of alternatives for each problem class experimented with correlated beliefs. 'OC' is the mean opportunity cost comparing tuned IE with others $OC^{\text{IE}} - OC^{\pi}$, with a positive value indicating a win for tuned IE. 'Prob.' is the probability that other policies outperform the tuned IE. We see from the table that z_α is highly problem dependent and the performance degrades quickly away from the optimal value. For some experimental applications, tuning can require running physical experiments, which may be very expensive or even entirely infeasible.

6.2. The issue of constructing priors

In MOLTE, we use MLE to fit the prior for test functions based on sampling measurements,

.

Problem Class	B	z^*_α	IE(1)		IE(2)		IE(3)		IE(4)		IE(5)	
			OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.	OC	Prob.
Bubeck4	I	2.086	0.002	0.40	0.001	0.45	0.002	0.46	0.015	0.47	0.017	0.47
Bubeck6	I	2.01	0.003	0.44	0.001	0.48	0.004	0.43	0.013	0.23	0.028	0.13
AUF_MNoise	I	1.1305	0.004	0.38	0.041	0.04	0.071	0.00	0.095	0.00	0.114	0.09
CamelBack	I	1.295	0.006	0.35	0.069	0.32	0.108	0.03	0.145	0.00	0.172	0.00
AUF_LNoise	C	0.9498	0.043	0.00	0.080	0.00	0.105	0.00	0.123	0.03	0.136	0.00
Branin	C	0.4438	0.001	0.25	0.005	0.32	0.014	0.07	0.023	0.01	0.032	0.01
Goldstein	C	0.079	0.071	0.00	0.090	0.00	0.101	0.00	0.108	0.00	0.113	0.00
Rosenbrock	C	0.9989	0.007	0.18	0.060	0.08	0.093	0.05	0.120	0.04	0.143	0.03

Table 8: Comparisons between tuned IE and IEs with fixed parameter values. The second column indicates the belief model, with I for independent belief and C for correlated belief. z^*_α is the tuned value for each problem class. The number included in the parenthesis is the parameter value used by each IE policy.

which seems like a tuning process. Yet designing a Bayesian prior is not necessarily the same as tuning parameters. In real world problems, such as applications in experimental sciences (although there are many other examples from other problem domains), the Bayesian prior may be based on an understanding of the physical system and might be based on the underlying chemistry/physics of the problem, a review of the literature, or past experience. This information might be qualitative in nature and is not easily incorporated by frequentist approaches. When this domain knowledgeable is available, and especially when experiments are expensive, Bayesian approaches are strongly preferred.

7. Conclusion

We offer MOLTE as a public-domain test environment to facilitate the process of more comprehensive comparisons, on a broader set of test problems and a broader set of policies, so that researchers can more easily draw insights into the behavior of different policies in the context of different problem classes. There has been a long history in the optimal learning literature of proving some sort of bound, supported at times by relatively thin empirical work by comparing a few policies on a small number of randomly generated problems. When choosing policies from a huge algorithms pool, we hope **MOLTE** can be a starting point for researchers, experimental scientists and students to more easily draw insights into the behavior of different policies in the context of different problem classes. We demonstrate the ability of **MOLTE** through extensive experimental results. We draw the conclusion that there is no universal best policy for all problem classes, and bounds, by themselves, do not provide reliable guidance to the policy that will work the best. We envision **MOLTE** as a modest spur to induce other researchers to come forward to study interesting questions involved in optimal learning, for example, the issue of tuning in this paper. We hope **MOLTE** can help with the current issue of relative paucity of empirical testing of learning algorithms.

References

Audibert JY, Bubeck S, *et al.* (2010). "Best arm identification in multi-armed bandits." *COLT 2010-Proceedings*.

Audibert JY, Munos R, Szepesvári C (2009). "Exploration-exploitation tradeoff using variance estimates in multi-armed bandits." *Theor. Comput. Sci.*, **410**(19).

Auer P, Cesa-Bianchi N, Fischer P (2002). "Finite-time analysis of the multiarmed bandit problem." *Machine learning*, **47**(2-3), 235–256.

Cappé O, Garivier A, Maillard OA, Munos R, Stoltz G, *et al.* (2013). "Kullback–leibler upper confidence bounds for optimal sequential allocation." *The Annals of Statistics*, **41**(3), 1516–1541.

Dixon L, Szegö G (1978). "The global optimization problem: an introduction." *Towards global optimization*, **2**, 1–15.

Frazier P, Powell W, Dayanik S (2009). "The knowledge-gradient policy for correlated normal beliefs." *INFORMS journal on Computing*, **21**(4), 599–613.

Frazier PI, Powell WB, Dayanik S (2008). "A knowledge-gradient policy for sequential information collection." *SIAM Journal on Control and Optimization*, **47**(5), 2410–2439.

Garivier A, Moulines E (2008). "On upper-confidence bound policies for non-stationary bandit problems." *arXiv preprint arXiv:0805.3415*.

Hu J, Fu MC, Marcus SI (2008). "A model reference adaptive search method for stochastic global optimization." *Communications in Information & Systems*, **8**(3), 245–276.

Huang D, Allen TT, Notz WI, Zeng N (2006). "Global optimization of stochastic black-box systems via sequential kriging meta-models." *Journal of global optimization*, **34**(3), 441–466.

Jones DR (2001). "A taxonomy of global optimization methods based on response surfaces." *Journal of global optimization*, **21**(4), 345–383.

Jones DR, Schonlau M, Welch WJ (1998). "Efficient global optimization of expensive black-box functions." *Journal of Global optimization*, **13**(4), 455–492.

Kaelbling LP (1993). *Learning in embedded systems*.

Kuleshov V, Precup D (2000). "Algorithms for multi-armed bandit problems." *Journal of Machine Learning Research*.

Lai TL, Robbins H (1985). "Asymptotically efficient adaptive allocation rules." *Advances in applied mathematics*, **6**(1), 4–22.

Martinez-Cantin R (2014). "BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits." *The Journal of Machine Learning Research*, **15**(1), 3735–3739.

Molga M, Smutnicki C (2005). "Test functions for optimization needs." *Test functions for optimization needs*.

Powell WB, Ryzhov IO (2012). *Optimal learning*, volume 841.

Ryzhov IO, Powell WB, Frazier PI (2012). "The algorithm for a general class of online learning problems." *Operations Research*, **60**(1), 180–195.

Snoek J, Larochelle H, Adams RP (2012). "Practical Bayesian optimization of machine learning algorithms." In *Advances in neural information processing systems*, pp. 2951–2959.

Srinivas N, Krause A, Kakade SM, Seeger M (2009). "Gaussian process optimization in the bandit setting: No regret and experimental design." *arXiv preprint arXiv:0912.3995*.

Thompson WR (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples." *Biometrika*, pp. 285–294.

Affiliation:

Yingfei Wang
Department of Computer Science
Princeton University
Princeton, NJ, 08540
E-mail: yingfei@cs.princeton.edu

Warren Powell
Department of Operations Research and Financial Engineering
Princeton University
Princeton, NJ, 08544
E-mail: powell@princeton.edu