Castle Labs

Department of Operations Research and Financial Engineering
Princeton University

MOLTE

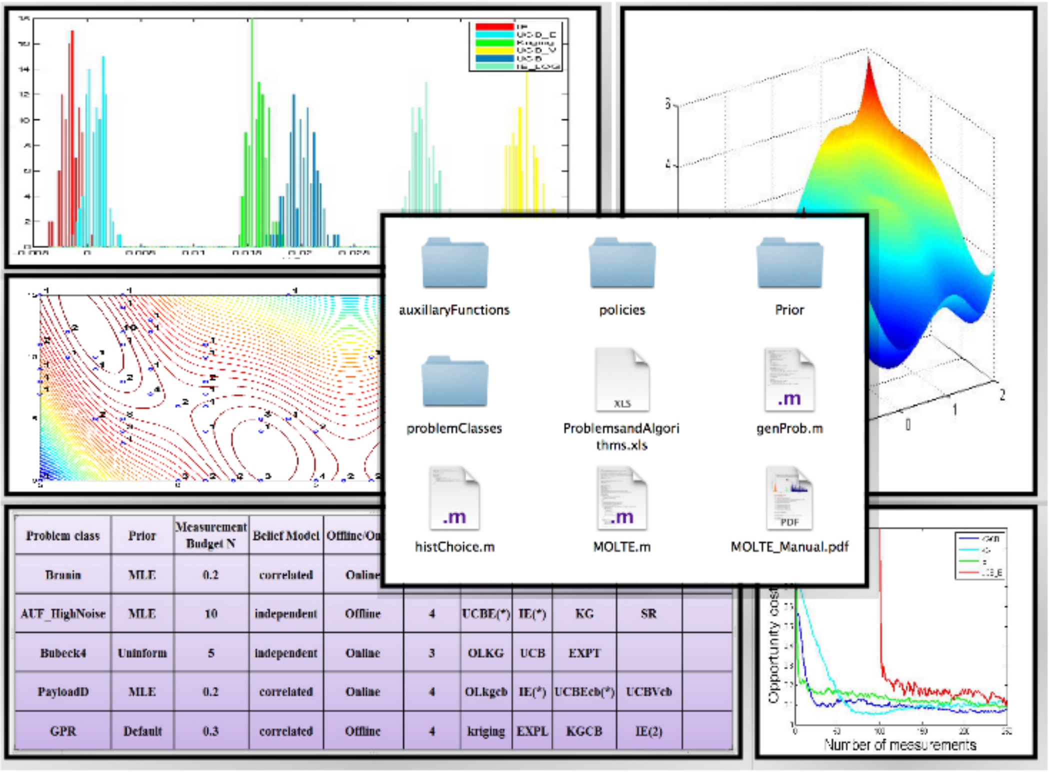
A modular, optimal learning testing environment

Yingfei Wang

Warren B. Powell

Robert Schapire

June 2, 2016



Problem class	Prior	Measurement Budget N	Belief Model	Offline/On						
Branin	MLE	0.2	correlated	Online						
AUF_HighNoise	MLE	10	independent	Offline	4	UCBE(*)	IE(*)	KG	SR	
Bubeck4	Uninform	5	independent	Online	3	OLKG	UCB	EXPT		
PayloadD	MLE	0.2	correlated	Online	4	OLkgcb	IE(*)	UCBEcb(*)	UCBVcb	
GPR	Default	0.3	correlated	Offline	4	kriging	EXPL	KGCB	IE(2)	

Description

MOLTE is a sequential design-of-experiments (stochastic optimization) testing environment for testing learning algorithms on a wide range of offline and online 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). The choice of problems and policies is guided through a spreadsheet-based interface. Users can follow the standard APIs to define a new problem class and new policy by writing a separate .m file.

Construction

MOLTE.m compares the polices specified in the Excel spreadsheet for each problem class for numP=100 times (which can be modified in MOLTE.m). Each time the simulator is run, it generates numTruth (which can be modified in MOLTE.m) different sample paths, shared between all the policies, computes the value of the objective function for each sample path and then averages the numTruth trials as the expected final reward or the expected cumulative rewards. The user may select to evaluate policies using either an online (“bandit”) objective function, or an offline objective function (ranking and selection, stochastic search).

Input Arguments

Spreadsheet: an Excel file (ProblemsandAlgorithms.xls) with each row a problem class with the specified policies under comparison. A possible spreadsheet is as follows:

Problem class	Prior	Measurement Budget	Belief Model	Offline/Online	Number of Policies				
PayloadD	MLE	0.2	independent	Offline	4	Kriging	EXPL	IE(1.7)	Thompson Sampling
Branin	MLE	10	correlated	Online	4	OLkgcb	UCBEcb(*)	IE(2)	BayesUCB
Bubeck4	uninformative	5	independent	Online	4	OLKG	UCB	SR	UCBV
GPR	Default	0.3	correlated	Offline	4	Kriging	kgcb	IE(*)	EXPT

For each problem, the following information has to be provided

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. If the problem has parameters, the user can use ( ) after the class name to input parameters; Otherwise use the default parameter value. For example, GPR(50, 0.45;100) specifies the value of the parameter for Gaussian Process Regression. Specifically, the prior mean is drawn from N(0, \sqrt(sigma)), the covariance matrix is of the form sigma*exp(-beta(x-x')). 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 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. **Uninformative** specifies zeros mean 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, e.g. there are 100 alternatives in the pre-coded Branin problem and the time horizon is specified to be 5*100 if this column is set to be 5.

Belief Model decides 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 final reward or the expected total rewards.

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 with respect to the entire table 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).

Output Data and Figures

All the data and figures are saved in a separate folder for each problem class. Within the folder of each problem class:

objectiveFunction.mat saves the value of the online or offline objective function achieved by each policy for each trial;

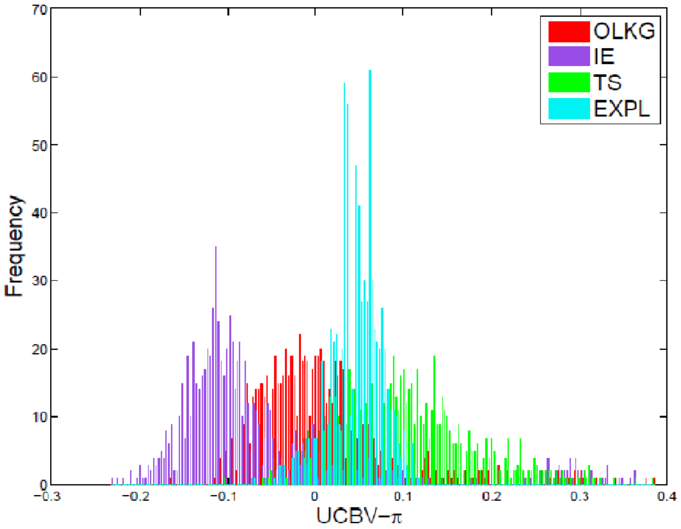
choice.mat saves the decisions made by each policy and the name of all policies;

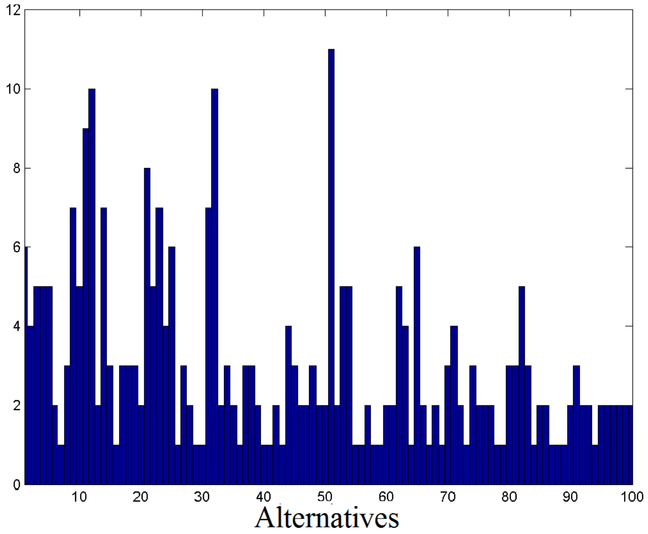
FinalFit.mat saves the final estimate of the surface by each policy after the measurement budget exhausted, together with the corresponding truth.

alpha.txt saves the value of tunable parameter for each policy that requires tuning;

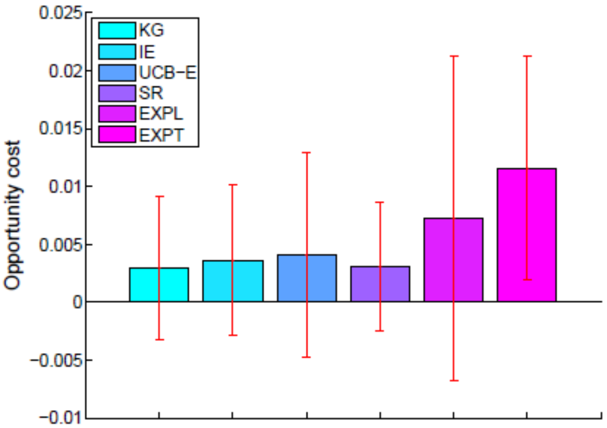
offline_hist.pdf is the histogram for each policy describing the distribution of the expected final reward compared to the reward obtained by the reference policy (whichever policy that is listed as the first policy in the input spreadsheet);
online_hist.pdf is the histogram describing the distribution of the expected total reward; e.g. the following left figure is obtained for online Bubeck4. A distribution centered around a positive value implies the policy underperforms KG;

histChoice.m can read in the **choice.mat** and generate the distribution of the chosen alternatives for each policy and each trial. e,g, the right figure shows the frequency of choosing each of the 100 alternatives with a measurement budget of 300.

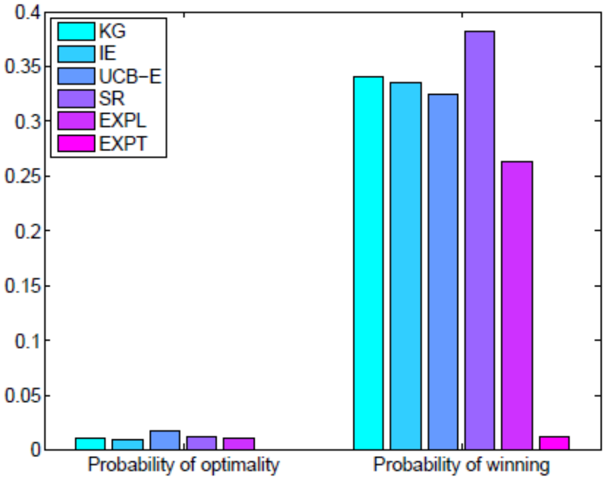




genProb.m can read in the **objectiveFunction.mat** and depict the mean opportunity cost with error bars indicating the standard deviation of each policy (in OC_hist.tif) as shown in the following left figure , together with the probability of each policy being optimal and being the best in the following right figure:

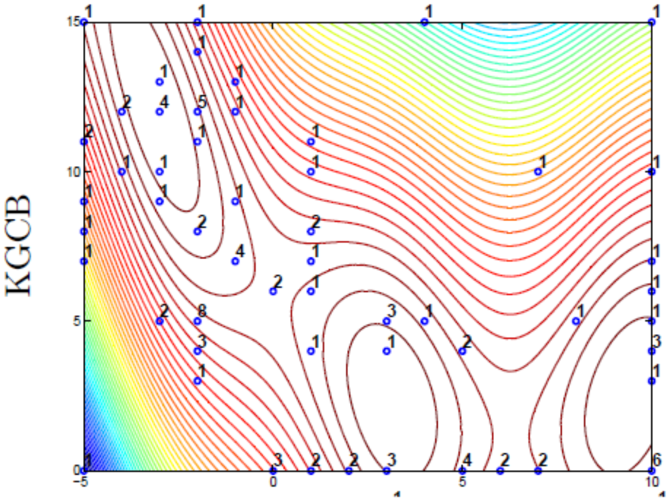


Goldstein: Opportunity cost

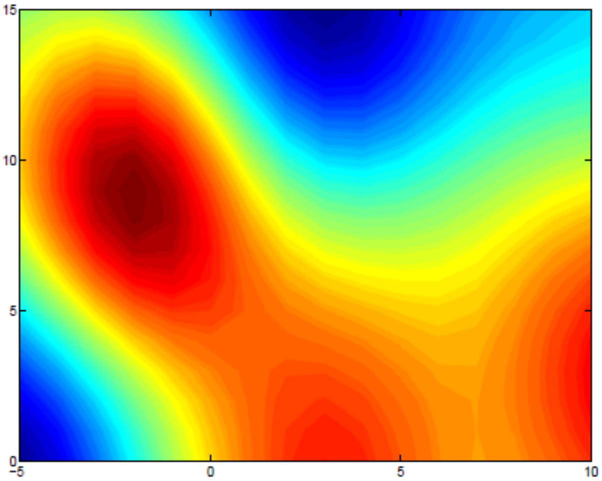


Goldstein:Probability of optimality/winning

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 **FinalFit.mat** and the number of times each policy sample each alternative (sampling distribution) stored in **choice.mat** to generate the following 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**:



Left column: sampling distribution.



Right column: posterior distribution.

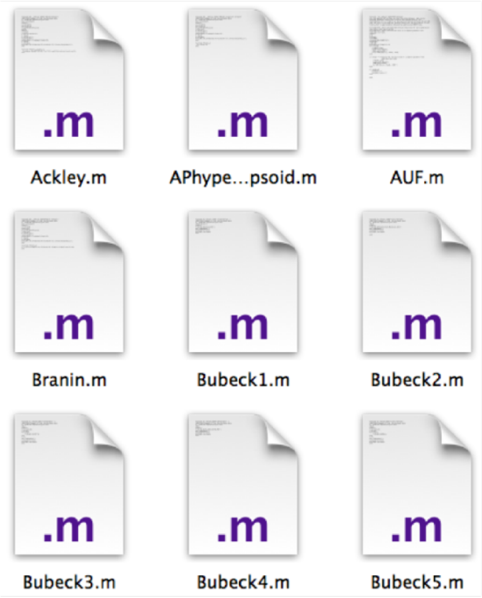
Add in new problem classes and/or policies

New problem class

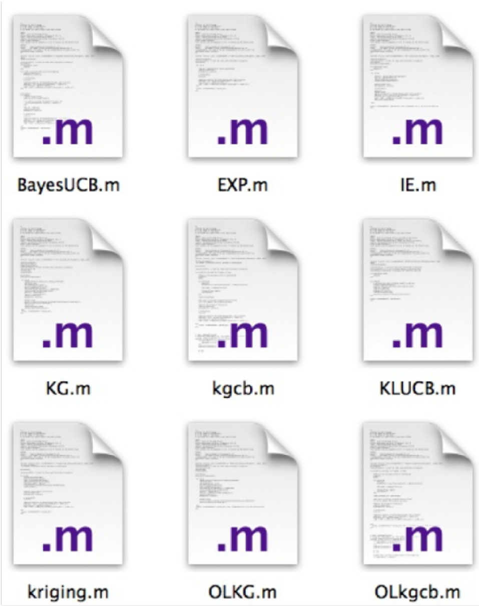
Each of the problems classes is organized in its own .m file in the ./problemClasses folder. The standard API is defined as Name(varargin), where varargin is used to pass input parameters with variable lengths for the problem class if needed.

New policy

Each of the problems classes is organized in its own .m file in the ./policies folder. The standard API is defined as KG(mu_0,beta_W,covM,samples, alpha, tune) where mu_0 and covM is specifies the prior distribution, beta_W is the known measure noise, samples are pre-generated and shared among all the policies, alpha is the tunable parameter and tune specifies whether tune this policy or use default value.



Problem Classes



Policies

Pre-coded Problem Classes

Synthetic test functions:

- Bubeck1~Bubeck7
- Asymmetric Unimodular Functions with the parameter chosen to be 0.2,0.5 and 0.8 with high or medium noise level, respectively
- Rosenbrock function with additive noise
- Pinter's function with additive noise
- Goldstein function with additive noise
- Griewank function with additive noise
- Branin’s function with additive noise
- Axis parallel hyper-ellipsoid function with additive noise
- Rastrigin’s function with additive noise
- Ackley’s function with additive noise
- Six-hump camel back function with additive noise
- Easom function with additive noise

Truth-From-Prior experiments:

- Gaussian Process Regression (with a default prior)

Parameterized families

- General AUF problems represented by one parameter drawn from U[0,1]

Real world applications:

- Payload delivery problem
- Immobilized nanoparticles design (with a default prior)

Pre-coded Policies

- Interval Estimation (IE) (can be used for correlated beliefs)
- Kriging (can be used for correlated beliefs)
- UCB (and a modified version UCBcb incorporating correlated beliefs)
- UCBNormal
- UCB-E (and a modified version UCBEcb incorporating correlated beliefs)
- UCB-V(and a modified version UCBVcb incorporating correlated beliefs)
- Bayes-UCB (can be used for correlated beliefs)
- KL-UCB
- Knowledge gradient policy for offline learning (can be used for correlated beliefs)
- Knowledge gradient for online learning (can be used for correlated beliefs)
- Successive rejects
- Thompson sampling (can be used for correlated beliefs)
- Pure exploration (can be used for correlated beliefs)
- Pure exploitation (can be used for correlated beliefs)