A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Eric Brochu, Mike Cora and Nando de Freitas

August 11, 2009

Abstract

We present a tutorial on Bayesian optimization, a method of finding the maximum of expensive cost functions. Bayesian optimization employs the Bayesian technique of setting a prior over the objective function and combining it with evidence to get a posterior function. This permits a utility-based selection of the next observation to make on the objective function, which must take into account both exploration (sampling from areas of high uncertainty) and exploitation (sampling areas likely to offer improvement over the current best observation). We also present two detailed applications of Bayesian optimization, with experiments – active user modelling with preferences, and hierarchical reinforcement learning. While the most common prior for Bayesian optimization is a Gaussian process, we also present Random forests as an example of an alternative prior.

1 Introduction

An enormous body of scientific literature has been devoted to the problem of optimizing a nonlinear function $f(\mathbf{x})$ over a compact set. In the realm of optimization, this problem is formulated concisely as follows:

$$\max_{\mathbf{x} \in A \subset \mathbb{R}^d} f(\mathbf{x})$$

There, one typically assumes that the *objective* function $f(\mathbf{x})$ has a know mathematical representation or, at least, that it is easy to evaluate. Despite the large influence of classical optimization in machine learning, many learning problems do not conform to these strong assumptions. Often, evaluating the objective function is expensive or even impossible.

In realistic sequential decision making problems, for example, one can only hope to obtain an estimate of the objective function by simulating future scenarios. Whether one adopts simple Monte Carlo simulation or adaptive schemes,

as proposed in the fields of planning and reinforcement learning, the process of simulation is invariably expensive. Moreover, in some applications such as experimental design and active sensing, the simulations correspond to expensive experiments: drug trials, destructive tests and financial investments. In active user modeling, $\mathbf{x}$ will typically represent attributes of a question being asked of a human by a computer, and $f(\mathbf{x})$ requires feedback from the human. Computers must ask the right questions and these questions must be kept to a minimum so as to avoid annoying the user.

Stochastic approximation is a popular idea for optimizing unknown objective functions [Kushner and Yin, 1997]. It plays an important role in machine learning. It is the core idea in most reinforcement learning algorithms [Bertsekas and Tsitsiklis, 1996; Sutton, 1998], learning methods for Boltzmann machines and deep belief networks [Younes, 1989; Hinton and Salakhutdinov, 2006] and parameter estimation for nonlinear state spaces models [Poyiadjis *et al.*, 2005; Martinez-Cantin *et al.*, 2006]. Unfortunately, stochastic approximation typically requires many function evaluations and is thus inappropriate for many of the tasks discussed in this tutorial.

Here, we will consider a suitable alternative known as *Bayesian optimization*. The goal of Bayesian optimization is to find the maximum of an objective function with as few function evaluations as possible, and as such Bayesian optimization techniques are some of the most efficient approaches in terms of the number of function evaluations required [Kushner, 1964; Mockus *et al.*, 1978; Locatelli, 1997; Jones *et al.*, 1998; Jones, 2001; Lizotte, 2008]. Much of this efficiency stems from the ability of Bayesian optimization schemes to trade-off exploration and exploitation of the search space to find multiple maxima.

In Bayesian optimization, one places a prior distribution over the space of objective functions. Since the objective function is very expensive to evaluate, it may be regarded as unknown for practical purposes. However, it is reasonable to assume that we have prior knowledge about some of its properties, such as smoothness or the potential location of its change-points. As we accumulate observations $\mathcal{D}_{1:n} = \{\mathbf{x}_i, f(\mathbf{x}_i)\}|_{i=1}^n$, the prior distribution is combined with the likelihood function $P(\mathcal{D}_{1:n}|f)$ to obtain the posterior distribution:

$$P(f|\mathcal{D}_{1:n}) \propto P(\mathcal{D}_{1:n}|f)P(f).$$

The posterior captures our updated beliefs about the unknown objective function. One may also interpret this step of Bayesian optimization as estimating the objective function with a surrogate function – typically the mean function of Gaussian process (later, we will introduce an alternative based on random forests). The surrogate function is often referred to as a response surface.

Bayesian optimization uses the surrogate to decide what the next value of the independent variable $\mathbf{x}$ should be. The decision represents an automatic trade-off between exploration (where the objective function is very uncertain) and exploitation (trying values of $\mathbf{x}$ where the objective function is expected to be high). This global optimization technique has the nice property that it aims to minimize the number of objective function evaluations. Moreover, it is likely

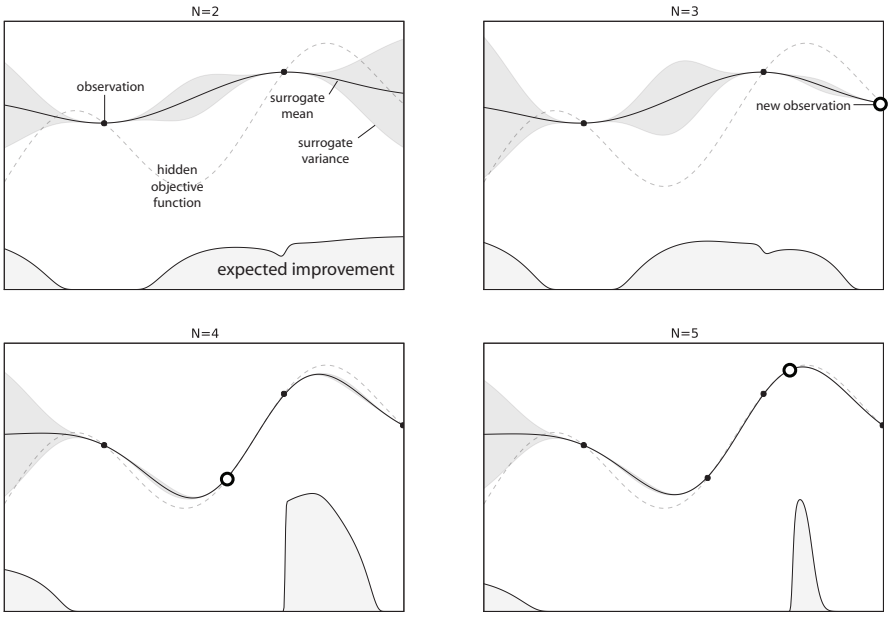


Figure 1: *An example of using Bayesian optimization on a toy 1D design problem. The figures show a Gaussian process (GP) approximation of the objective function over four iterations of sampled values of the objective function. The figure also shows the expected improvement of each potential next sampling location in the lower shaded plots. The expected improvement is high where the GP predicts a high objective (exploitation) and where the prediction uncertainty is high (exploration) – areas with both attributes are sampled first. Note that the area on the far left remains unsampled, as while it has high uncertainty, it is (correctly) predicted to offer little improvement over the highest observation.*

to do well even in settings where the objective function has many local maxima. Figure 1 shows a typical run of Bayesian optimization on a 1D problem.

In much of the literature – including this paper – the Bayesian optimization problem is cast as one of global optimization. However, this is not entirely accurate. Even in a noise-free domain, evaluating an objective function with Lipschitz continuity l, guaranteeing the best observation $f(\mathbf{x}^{best}) \geq \max_x f(x) - \epsilon$ requires sampling with sufficient density to ensure that no point in the space is more than distance $\frac{\epsilon}{l}$ from the nearest sample. This is not practical for many real-world problems, particularly ones where dimensionality is high and function evaluations are expensive. The goal, instead, is to use evidence and prior knowledge to maximize the posterior at each step, so that each new evaluation decreases the distance between the true global maximum and the predicted maximum, an activity sometimes called "realistic" or "practical" optimization.

1.1 Overview

In Section 2, we give an overview of the Bayesian optimization approach and its history. In Section 3, we show how to set the prior over an objective function using Gaussian processes. Section 4 presents maximum expected utility – the method by which we determine the next point to evaluate.

The second half of the tutorial builds on the basic Bayesian optimization model. In Sections 5 and 6 we discuss extensions to Bayesian optimization for active user modelling in preference galleries, and hierarchical control problems, respectively. In Section 7, we discuss an example of an alternative function prior based on Random Forests.

Finally, we end the tutorial with a brief discussion of the pros and cons of Bayesian optimization in Section 8.

2 The Bayesian Optimization Approach

Assume we have placed a prior distribution over the objective function and that we are interested in deciding what the next query point $\mathbf{x}_1$ should be. Most classical optimization procedures do not use such a prior. Instead, they choose the query point that minimizes the maximal deviation of $f(\mathbf{x}_1)$ from the global maximum $f(\mathbf{x}^\star)$. As mentioned in the previous section, this worst-case (minmax) approach leads to exponential costs in computation. Bayesian optimization, on the other hand, uses the prior so as to minimize the expected deviation [Mockus *et al.*, 1978]:

$$\mathbf{x}_1 = \arg\min_x \int \|f(\mathbf{x}) - f(\mathbf{x}^\star)\| dP(f).$$

That is, since we don't know the function f, we integrate over it. This average-case approach has weaker demands on computation than the worst-case approach. As a result, it may provide faster solutions in many practical domains where one does not believe the worst-case scenario is very plausible.

Bayesian optimization follows the principle of *maximum expected utility* (equivalently, minimum expected risk), with the risk function corresponding to the deviation of the function from the maximum. This principle arises as a rational consequence of fairly unassailable axioms in game theory [von Neumann and Morgenstern, 1947]. An agent expecting to behave optimally must maximize its expected utility; see [Shoham and Leyton-Brown, 2009] for a more comprehensive treatment.

Bayesian optimization involves three stages: defining the prior distribution over functions, updating this distribution using Bayes' rule and deciding what values of $\mathbf{x}$ to sample next. The procedure is outlined in Algorithm 1. The process of deciding where to sample next requires the choice of a utility function and a way of optimizing the expectation of this utility (often referred to as the *acquisition function, infill function* or *expected improvement function*) with respect to the posterior distribution of the objective function. This secondary

optimization problem is often easier because the expected utility is typically chosen so that it is easy to evaluate. Moreover, in practice one just needs to find a reasonable peak of the expected utility.

Algorithm 1 Bayesian Optimization

1: $n = 1$
2: **while** samples available **do**
3: Update the expressions for the sufficient statistics of the posterior distribution (*e.g.* mean and variance of a Gaussian process) using the available data $\mathcal{D}_{1:n}$.
4: Find $\mathbf{x}_{n+1}$ by minimizing the expected deviation from the maximum. The expectation is taken with respect to the posterior distribution $P(f|\mathcal{D}_{1:n})$.
5: Evaluate the objective function, $f(\mathbf{x}_{n+1})$
6: Augment the data $\mathcal{D}_{1:n+1} = \{\mathcal{D}_{1:n}, (\mathbf{x}_{n+1}, f(\mathbf{x}_{n+1}))\}$
7: $n = n + 1$
8: **end while**

2.1 Brief history

The earliest work we are aware of resembling the modern Bayesian optimization approach is the early work of Kushner [1962; 1964], who used Wiener processes for one-dimensional problems. Kushner's decision model was based on maximizing the probability of improvement, and included a parameter that controlled the trade-off between 'more global' and 'more local' optimization, in the same spirit as the exploration-exploitation trade-off. Later work extended Kushner's technique to multidimensional optimization, using, for example, interpolation in a Delauney triangulation of the space [Elder, 1992] or line search methods [Stuckman, 1988].

Meanwhile, in the former Soviet Union, Močkus and colleagues developed a multidimensional Bayesian optimization method using linear combinations of Wiener fields, some of which was published in English as [Mockus *et al.*, 1978]. This paper also, significantly, describes an acquisition function that is based on myopic expected improvement of the posterior, which has been widely adopted in Bayesian optimization as the expected improvement function. A more recent review of Močkus' approach is [Mockus, 1994]. The method was also developed in parallel in the geophysics field, where Gaussian processes are referred to as *Kriging*, in honour of a South African student who developed this technique for this Masters thesis at the University of the Witwatersrand [Krige, 1951].

More recently, Bayesian optimization using Gaussian processes has been successfully applied to derivative-free optimization and experimental design [Jones *et al.*, 1998], where it is called Efficient Global Optimization, or *EGO*. Since EGO's publication, there has evolved a body of work devoted to extending the algorithm, particularly in adding constraints to the optimization problem [Schonlau *et al.*, 1998; Audet *et al.*, 2000; Sasena, 2002; Boyle, 2007], and in modelling noisy functions [Bartz-Beielstein *et al.*, 2005; Huang *et al.*, 2006; Hutter *et al.*, 2009].

There exist several consistency proofs for this algorithm in the one-dimensional setting [Locatelli, 1997] and one for a simplification of the algorithm using sim-

plicial partitioning in higher dimensions [Žilinskas and Žilinskas, 2002]. The convergence of the algorithm using multivariate Gaussian processes has been recently established in [Vasquez and Bect, 2008].

Bayesian optimization has recently begun to appear in the machine learning literature, in particular in robotics, active learning, reinforcement learning and preference learning applications [Lizotte *et al.*, 2007; Martinez-Cantin *et al.*, 2007; Brochu *et al.*, 2007; Cora, 2008; De Grave *et al.*, 2008; Martinez-Cantin *et al.*, 2009]. The approach in [Brochu *et al.*, 2007] seems to be the first to treat non-Gaussian observations. The PhD thesis of Lizotte [2008] is an excellent reference on the topic.

3 Priors Over Functions

Any Bayesian method depends on a prior distribution, by definition. Assume we are given an objective function, $f(\mathbf{x}), \mathbf{x} \in A \subset \mathbb{R}^d$. A Bayesian optimization method will converge to the global maximum, provided the prior converges to the maximum as N becomes large. This is so if the conditional variance converges to zero if and only if the distance from the nearest observation approaches zero [Mockus, 1994]. We often further restrict this to continuous functions, but this is not strictly necessary: we can use this technique to search a finite set of samples, for instance, as long as they observe this convergence prior.

Many models could be used for this prior, and in Section 7 we will look at an interval-continuous model based on Random Forests. However, the Bayesian optimization literature to date has focused almost exclusively on the Gaussian process (GP). A GP is a stochastic process for which any finite combination of samples will be normally distributed[1]. A stochastic process is the counterpart to a deterministic process: in the latter, we can get a specific response for a given input, but in a stochastic process, we get a distribution over the possible responses. For our purposes, a GP can be thought of as analogous to a function, but instead of returning a scalar $f(\mathbf{x})$ for an arbitrary $\mathbf{x}$, it returns the mean and variance of a Normal distribution (Figure 2) over the possible values of $f(\mathbf{x})$. It's no accident that GPs were first widely used in geostatistics. If a mining company is deciding where to dig based on mineral samples of an area, it requires an idea of the range of possible outcomes, not an interpolation with no concept of risk.

Just as a Gaussian distribution is a distribution over a random variable, completely specified by its mean and covariance, a GP is a distribution over functions, completely specified by its mean function, $m(\cdot)$ and covariance function, $\mathbf{K}(\cdot, \cdot)$:

$$f(\cdot) \sim GP(m(\cdot), \mathbf{K}(\cdot, \cdot)).$$

For convenience, we assume here that the mean is the zero function, but if

[1] It is this property that is the origin of the name – Carl Friederich Gauss never studied GPs.

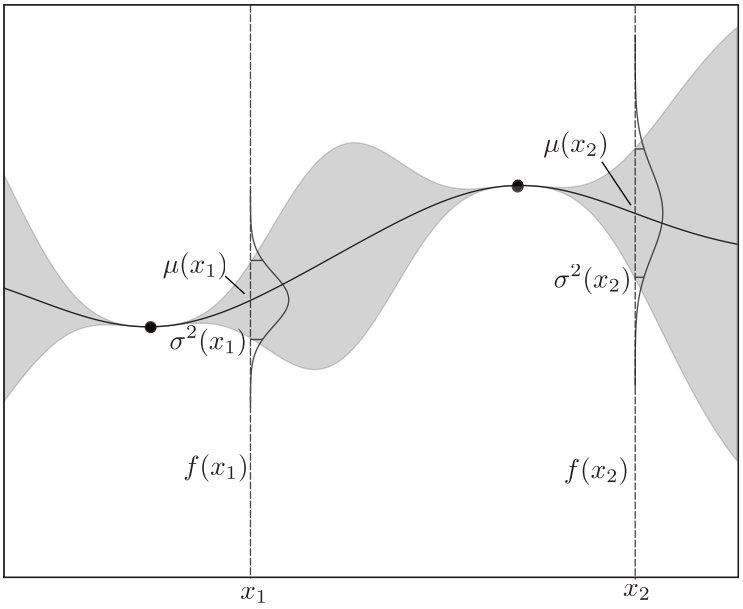


Figure 2: *Simple 1D Gaussian process with two observations. The superimposed Gaussians correspond to the GP mean $\mu(\cdot)$ and variance $\sigma^2(\cdot)$ (depicted with 95% confidence intervals) of the predictions at points x_1 and x_2. The curve is the GP surrogate mean prediction of the objective function using two data. Given a new sample, say x_1 one samples the function value $f(x_1)$ from the Gaussian $\mathcal{N}(\mu, \sigma^2)$.*

desired, we could learn it using the more general method described by Martinez-Cantin *et al* [2007]. This leaves us the more interesting question of defining the covariance function $\mathbf{K}(\cdot, \cdot)$. A very popular choice is the squared exponential function, also known as a radial basis function, or Gaussian[2] covariance function:

$$k(\mathbf{x}_j, \mathbf{x}_k) = \exp\left(-\frac{1}{2}\left|\mathbf{x}_j - \mathbf{x}_k\right|^2\right). \tag{1}$$

Note that this function approaches 1 as values get close together and 0 as they get further apart. Two points that are close together can be expected to have a very large influence on each other, whereas distant points have almost none. This is not true of all covariance functions, but it is often desirable in a Bayesian optimization setting.

We can sample the Gaussian process at n points by choosing the indices $\{\mathbf{x}_{1:n}\}$ and sampling the values of the function at these indices to produce the pairs $\{\mathbf{x}_{1:n}, \mathbf{f}_{1:n}\}$, where $\mathbf{f}_{1:n} = f(\mathbf{x}_{1:n})$. The function values are drawn according to a multivariate Normal distribution $N(\mathbf{m}_{1:n}, \mathbf{K})$, where the mean vector is

[2]Like Gauss himself, this has nothing to do with Gaussian processes being Gaussian, and everything to do with the importance of the Gaussian distribution to statistics.

$\mathbf{m}_{1:n} = m(\mathbf{x}_{1:n})$ and the kernel matrix is given by:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}. \tag{2}$$

Of course, the diagonal values of this matrix are 1 (each point is perfectly correlated with itself), which is only possible in a noise-free environment. We will discuss noise in Section 3.2. Also, recall that we have for tutorial reasons chosen zero mean function.

Assume that we already have the observations $\{\mathbf{x}_{1:n}, \mathbf{f}_{1:n}\}$, say from previous iterations, and that we want to use Bayesian optimization to decide what point $\mathbf{x}_{n+1}$ should be considered next. Let us denote the value of the function at this arbitrary point as $f_{n+1} = f(\mathbf{x}_{n+1})$. Then, by the properties of Gaussian processes, $\mathbf{f}_{1:n}$ and f_{n+1} are jointly Gaussian:

$$\begin{bmatrix} \mathbf{f}_{1:n} \\ f_{n+1} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) \end{bmatrix}\right),$$

where

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}_{n+1}, \mathbf{x}_1) & k(\mathbf{x}_{n+1}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{n+1}, \mathbf{x}_n) \end{bmatrix}$$

Hence, using the rules of conditioning for Gaussian distributions and some tedious algebra, one can easily arrive at an expression for the predictive posterior distribution:

$$p(f_{n+1}|\mathbf{f}_{1:n}) = \mathcal{N}(\mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}_{1:n}, k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k})$$

That is, for any point $\mathbf{x}_{n+1}$, the posterior distribution is fully characterized by the following pair of statistics:

$$\begin{aligned} \mu(\mathbf{x}_{n+1}) &= \mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}_{1:N} \\ \sigma^2(\mathbf{x}_{n+1}) &= k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}. \end{aligned}$$

As long as the number of query points is small, the GP predictions are very easy to compute.

3.1 Choice of covariance functions

The squared exponential kernel above is actually a little naive, in that divergences of all features of $\mathbf{x}$ affect the covariance equally. This might not correspond to observed behaviour very well – we might observe a very high covariance on one feature and virtually none on another. To account for this, we add a *hyperparameter* vector $\theta \in \mathbb{R}^d$ of length-scales, which essentially control how far you need to travel along each axis for correlation to fall off (other formulations of the hyperparameters are possible):

$$k(\mathbf{x}_j, \mathbf{x}_k) = \exp\left(-\frac{1}{2}(\mathbf{x}_j - \mathbf{x}_k)^T \mathrm{diag}(\theta)^{-2}(\mathbf{x}_j - \mathbf{x}_k)\right)$$

In many applications, θ can be learned by maximum a posteriori inference, but in active learning it is common to start with very few data points, which is known to result in very poor kernel parameters when using MAP. Two common work-arounds exist. In the experimental design literature, Bayesian optimization is often initializes using Latin hypercubes (*eg* [Santner *et al.*, 2003]), which bypasses the need to learn parameters until enough data has been selected to make MAP feasible. However, the number of data required is large – typically $10d$ – which are selected without using information about function evaluations. This is not always efficient enough for active learning scenarios, and machine learning approaches have tended to use informative priors on the parameters, or rely on an expert's selection of an initial smoothing kernel width. As more data is collected, the initial prior can be replaced with MAP.

Another import kernel for Bayesian optimization is the Matérn kernel, which incorporates a smoothness parameter ν to permit greater flexibility in modelling functions:

$$k(\mathbf{x}_j, \mathbf{x}_k) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(2\sqrt{\nu}|\mathbf{x}_j - \mathbf{x}_k|\right)^{\nu} H_{\nu}\left(2\sqrt{\nu}|\mathbf{x}_j - \mathbf{x}_k|\right),$$

where $\Gamma(\cdot)$ and $H_{\nu}(\cdot)$ are the Gamma function and the Bessel function of order ν. Note that as $\nu \to \infty$, the Matérn kernel reduces to the squared exponential kernel, and when $\nu = 0.5$, it reduces to the unsquared exponential kernel. As with the squared exponential, length-scale hyperparameter are often incorporated.

While the squared exponential and Matérn are the most common kernels for GPs, numerous others have been examined in the machine learning literature. See, *eg*, [Genton, 2001] for an overview.

3.2 Observation Models

The model we've used so far assumes that we have perfectly noise-free observations. In real life, this is rarely possible, and instead of observing $f(\mathbf{x})$, we can often only observe a noisy transformation of $f(\mathbf{x})$.

3.2.1 Gaussian observations

The simplest transformation arises when $f(\mathbf{x})$ is corrupted with i.i.d., zero mean, Gaussian noise ϵ with variance σ_n^2. If the noise is additive, we can easily add the noise distribution $\mathcal{N}(0, \sigma_n^2)$ to the Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{K})$. Since the mean is zero, this type of noise simply requires that we replace the kernel $\mathbf{K}$ with the following kernel for the noisy observations of $f(\cdot)$:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} + \sigma_n^2 I$$

3.2.2 Probit model for binary observations

The *probit model* allows us to deal with binary observations of $f(\cdot)$ in general. That is, every time we try a value of $\mathbf{x}$, we get back a binary variable, say either zero or one. From the binary observations, we have to infer the latent function $f(\cdot)$. In order to marry the presentation in this section to the user modeling applications discussed later in this tutorial, we will introduce probit models in the particular case of preference learning.

Assume we have shown the user M pairs of items. In each case, the user has chosen which item she likes best. The data set therefore consists of the ranked pairs:

$$\mathcal{D} = \{\mathbf{r}_k \succ \mathbf{c}_k; \;\; k = 1, \ldots, M\},$$

where the symbol $\succ$ indicates that the user prefers $\mathbf{r}$ to $\mathbf{c}$. We use $\mathbf{x}_{1:n}$ to denote the n distinct elements in the training data. That is, $\mathbf{r}_k$ and $\mathbf{c}_k$ correspond to two elements of $\mathbf{x}_{1:n}$. Note that we can interpret $\mathbf{r}_k \succ \mathbf{c}_k$ as a binary variable, say z, that takes value 1 when $\mathbf{r}_k$ is preferred to $\mathbf{c}_k$ and is 0 otherwise.

Later, when we do Bayesian optimization, our goal will be to compute the item $\mathbf{x}$ (not necessarily in the training data) with the highest user valuation in as few comparisons as possible. For the time being, we focus on deriving the expression for the predictive posterior distributions of the Gaussian process with binary observations (discrete choices in this case).

In the probit approach, we model the valuation functions $u(\cdot)$ for items $\mathbf{r}$ and $\mathbf{c}$ as follows:

$$\begin{aligned} u(\mathbf{r}_k) &= f(\mathbf{r}_k) + e_{rk} \\ u(\mathbf{c}_k) &= f(\mathbf{c}_k) + e_{ck}, \end{aligned} \tag{3}$$

where the noise terms are Gaussian: $e_{rk} \sim \mathcal{N}(0, \sigma^2)$ and $e_{ck} \sim \mathcal{N}(0, \sigma^2)$. Following [Chu and Ghahramani, 2005b], we assign a nonparametric Gaussian process prior to the unknown mean valuation: $f(\cdot) \sim GP(0, K(\cdot, \cdot))$. That is, at the n training points:

$$p(\mathbf{f}) = |2\pi\mathbf{K}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}\right),$$

where $\mathbf{f} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)\}$ and the symmetric positive definite covariance $\mathbf{K}$ has entries (kernels) $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. One can learn the parameters of the kernel via maximum likelihood, but in the context of Bayesian optimization, this is unsound due to the scarcity of data. To remedy this, one can use subjective priors using simple heuristics, such as expected dataset spread.

Random utility models such as (3) have a long and influential history in psychology and the study of individual choice behaviour in economic markets. Daniel McFadden's Nobel Prize speech [McFadden, 2001] provides a glimpse of this history. Many more comprehensive treatments appear in classical economics books on discrete choice theory.

Under our Gaussian utility models, the probability that item $\mathbf{r}$ is preferred

to item $\mathbf{c}$ is given by:

$$\begin{aligned} P(\mathbf{r}_k \succ \mathbf{c}_k | f(\mathbf{r}_k), f(\mathbf{c}_k)) &= P(u(\mathbf{r}_k) > u(\mathbf{c}_k) | f(\mathbf{r}_k), f(\mathbf{c}_k)) \\ &= P(e_{ck} - e_{rk} < f(\mathbf{r}_k) - f(\mathbf{c}_k)) \\ &= \Phi\left[\frac{f(\mathbf{r}_k) - f(\mathbf{c}_k)}{\sqrt{2}\sigma}\right], \end{aligned} \tag{4}$$

where $\Phi(\cdot)$ is the CDF of the standard Normal distribution. This model, relating binary observations to a continuous latent function, is known as the Thurstone-Mosteller law of comparative judgement [Thurstone, 1927; Mosteller, 1951]. In statistics it goes by the name of binomial-probit regression. Note that one could also easily adopt a logistic (sigmoidal) link function $\varphi(d_k) = (1 + \exp(-d_k))^{-1}$. In fact, such choice is known as the Bradley-Terry model [Stern, 1990]. If the user had more than two choices one could adopt a multinomial-probit model. This multi-category extension would, for example, enable the user to state no preference for any of the two items being presented.

Our goal is to estimate the posterior distribution of the latent utility function given the discrete data. That is, we want to compute:

$$p(\mathbf{f}|\mathcal{D}) \propto p(\mathbf{f}) \prod_{k=1}^{M} p(d_k|\mathbf{f}),$$

where $d_k = \frac{f(\mathbf{r}_k) - f(\mathbf{c}_k)}{\sqrt{2}\sigma}$. Although there exist sophisticated variational and Monte Carlo methods for approximating this distribution, we favour a simple strategy: Laplace approximation [Chu and Ghahramani, 2005b]. Our motivation for doing this is the simplicity and computational efficiency of this technique. Moreover, given the amount of uncertainty in user valuations, we believe the choice of approximating technique plays a small role and hence we expect the simple Laplace approximation to perform reasonably in comparison to other techniques.

The Laplace approximation follows from Taylor-expanding the log-posterior about a set point $\widehat{\mathbf{f}}$:

$$\log p(\mathbf{f}|\mathcal{D}) = \log p(\widehat{\mathbf{f}}|\mathcal{D}) + \mathbf{g}^T(\mathbf{f} - \widehat{\mathbf{f}}) - \frac{1}{2}(\mathbf{f} - \widehat{\mathbf{f}})\mathbf{H}(\mathbf{f} - \widehat{\mathbf{f}}),$$

where $\mathbf{g} = \nabla_{\mathbf{f}} \log p(\mathbf{f}|\mathcal{D})$ and $\mathbf{H} = -\nabla_{\mathbf{f}}\nabla_{\mathbf{f}} \log p(\mathbf{f}|\mathcal{D})$. At the mode of the posterior ($\widehat{\mathbf{f}} = \mathbf{f}^{MAP}$), the gradient $\mathbf{g}$ vanishes, and we obtain:

$$p(\mathbf{f}|\mathcal{D}) \approx p(\widehat{\mathbf{f}}|\mathcal{D}) \exp\left[-\frac{1}{2}(\mathbf{f} - \widehat{\mathbf{f}})\mathbf{H}(\mathbf{f} - \widehat{\mathbf{f}})\right]$$

In order to obtain this approximation, we need to compute the maximum a posteriori (MAP) estimate $\mathbf{f}^{MAP}$, the gradient $\mathbf{g}$ and the information matrix $\mathbf{H}$.

The gradient is given by:

$$\begin{aligned}\mathbf{g} &= \nabla_{\mathbf{f}} \log p(\mathbf{f}|\mathcal{D}) \\ &= \nabla_{\mathbf{f}} \left[const - \frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} + \sum_{k=1}^{M} \log \Phi(d_k) \right] \\ &= -\mathbf{K}\mathbf{f} + \nabla_f \left[\sum_{k=1}^{M} \log \Phi(d_k) \right] = -\mathbf{K}\mathbf{f} + \mathbf{b},\end{aligned}$$

where the i-th entry of the N-dimensional vector $\mathbf{b}$ is given by:

$$b_i = \frac{1}{\sqrt{2}\sigma} \sum_{k=1}^{M} \frac{\phi(d_k)}{\Phi(d_k)} \left[\frac{\partial}{\partial f(\mathbf{x}_i)} (f(\mathbf{r}_k) - f(\mathbf{c}_k)) \right],$$

where $\phi(\cdot)$ denotes the PDF of the standard Normal distribution. Clearly, the derivative $\alpha_k(\mathbf{x}_i) = \frac{\partial}{\partial f(\mathbf{x}_i)}(f(\mathbf{r}_k) - f(\mathbf{c}_k))$ is 1 when $\mathbf{x}_i = \mathbf{r}_k$, -1 when $\mathbf{x}_i = \mathbf{c}_k$ and 0 otherwise. Proceeding to compute the second derivative, one obtains the Hessian: $\mathbf{H} = \mathbf{K}^{-1} + \mathbf{C}$, where the matrix $\mathbf{C}$ has entries

$$\begin{aligned}\mathbf{C}_{i,j} &= -\frac{\partial^2}{\partial f(\mathbf{x}_i)\partial f(\mathbf{x}_j)} \sum_{k=1}^{M} \log \Phi(d_k) \\ &= \frac{1}{2\sigma^2} \sum_{k=1}^{M} \alpha_k(\mathbf{x}_i)\alpha_k(\mathbf{x}_j) \left[\frac{\phi(d_k)}{\Phi^2(d_k)} + \frac{\phi^2(d_k)}{\Phi(d_k)} d_k \right]\end{aligned}$$

The Hessian is a positive semi-definite matrix [Chu and Ghahramani, 2005b]. Hence, one can find the MAP estimate with a simple Newton-Raphson recursion:

$$\mathbf{f}^{new} = \mathbf{f}^{old} - \mathbf{H}^{-1}\mathbf{g} \mid_{\mathbf{f}=\mathbf{f}^{old}}.$$

At $\mathbf{f} = \mathbf{f}^{MAP}$, we have

$$p(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}\left(\mathbf{K}^{-1}\mathbf{b}, (\mathbf{K}^{-1} + \mathbf{C})^{-1}\right).$$

The goal of our derivation, namely the predictive distribution $p(f_{n+1}|\mathcal{D})$, follows by straightforward convolution of two Gaussians:

$$\begin{aligned}p(f_{n+1}|\mathcal{D}) &= \int p(f_{n+1}|\mathbf{f}) p(\mathbf{f}|\mathcal{D}) d\mathbf{f} \\ &= \mathcal{N}(\mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}^{MAP}, k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T(\mathbf{K} + \mathbf{C}_{MAP}^{-1})^{-1}\mathbf{k}).\end{aligned}$$

3.2.3 Other observation models

One can extend the idea of using a latent Gaussian process beyond binary or Gaussian observations. The probit model can be easily generalized with multinomial observation functions. The same strategy might be applied to model other types of data, including, for example, count data (with Poisson distributions), heavy tailed positive measurements (with Gamma distributions) and variables in the interval $[0, 1]$ (with Dirichlet distributions). Transformations using warped Gaussian processes [Snelson *et al.*, 2003] could also be explored.

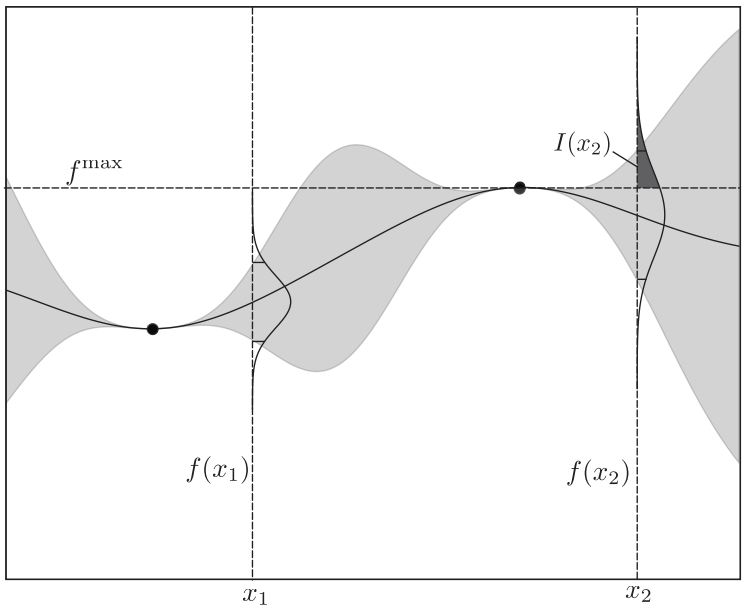


Figure 3: *Gaussian process from Figure 2, additionally showing the region of expected improvement. The maximum observation is $\mu_{\max}$. The darkly-shaded area in the superimposed Gaussian above the dashed line can be used as a measure of improvement, $I(x)$. The model predicts almost no possibility of improvement by observing at x_1, while sampling at x_2 is more likely to improve on $f^{\max}$.*

4 Maximum Expected Utility for Bayesian Optimization

Now that we know how to place priors over smooth functions and how to update these priors in light of new observations, we will focus our attention on the maximum expected utility component of Bayesian optimization. In particular, we want to minimize the expected deviation from the true maximum $f(\mathbf{x}^\star)$, when choosing a new trial point:

$$\begin{aligned}\mathbf{x}_{n+1} &= \arg\min_{\mathbf{x}} \mathbb{E}(\|f_{n+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| \mid \mathcal{D}_n) \\ &= \arg\min_{\mathbf{x}} \int \|f_{n+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| p(f_{n+1}|\mathcal{D}_n) df_{n+1},\end{aligned}$$

where $\mathcal{D}_n$ denotes all the data up to time n. Note that this decision process is myopic in that it only considers one-step-ahead choices. However, if we want to plan two-steps ahead, we can easily apply recursion:

$$\mathbf{x}_{n+1} = \arg\min_{\mathbf{x}} \mathbb{E}\left(\min_{\mathbf{x}'} \mathbb{E}(\|f_{n+2}(\mathbf{x}') - f(\mathbf{x}^\star)\| \mid \mathcal{D}_{n+1}) \mid \mathcal{D}_n\right)$$

One could continue applying this procedure of dynamic programming for as many steps ahead as desired. However, because of its expense, Močkus [1978] proposed the alternative of maximizing the expected improvement with respect to the best value $f^{\max}$ found thus far. Specifically, Močkus defined the improvement function as:

$$I(\mathbf{x}) = \max\{0, f_{n+1}(\mathbf{x}) - f^{\max}\}.$$

That is, $I(\mathbf{x})$ is positive when the prediction is higher that the best value known thus far. Otherwise, $I(\mathbf{x})$ is set to zero. The new query point is found by maximizing the expected improvement:

$$\mathbf{x} = \arg\max_{\mathbf{x}} \mathbb{E}(\max\{0, f_{n+1}(\mathbf{x}) - f^{\max}\} \mid \mathcal{D}_n)$$

Note that is is still a maximum expected utility expression. We have simply changed the utility function. The expected improvement can be evaluated analytically, see for example [Jones, 2001], yielding:

$$EI(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f^{\max})\Phi(Z) + \sigma^2(\mathbf{x})\phi(Z) & \text{if } \sigma^2(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma^2(\mathbf{x}) = 0 \end{cases} \tag{5}$$

where $Z = \frac{\mu(\mathbf{x}) - f^{\max}}{\sigma^s(\mathbf{x})}$ and $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard Normal distribution respectively. Figure 3 illustrates a typical expected improvement scenario.

The expected improvement function is the most popular way of querying a new point. Other forms based on tail-probabilities and entropy have also been proposed, but none have yet proved to supersede the good old expected improvement function of Močkus. It should be said however, that being myopic is not a requirement here. For example, it is possible to derive analytical expressions for the two-step ahead expected improvement [Ginsbourger *et al.*, 2008]. This is indeed a very promising recent direction.

4.1 Maximizing the expected improvement

To find the point at which to sample, we still need to maximize the constrained objective $EI(\mathbf{x})$. *Unlike the original unknown objective function, $EI(\cdot)$ can be cheaply sampled.* To optimize, we use *DIRECT* [Jones *et al.*, 1993], a deterministic, derivative-free optimizer. It uses the existing samples of the objective function to decide how to proceed to DIvide the feasible space into finer RECTangles. A particular advantage in active learning applications is that DIRECT can be implemented as an "any-time" algorithm, so that as long as the user is doing something else, it continues to optimize, and when interrupted, the program can use the best results found to that point in time. Methods such as sequential quadratic programming, branch-and-bound and quasi-Newton hill climbers have also been used, and perform similarly.

4.2 Exploration-exploitation trade-off

The expectation of the improvement function with respect to the predictive distribution of the Gaussian process enables us to balance the trade-off of exploiting and exploring. When exploring, we should choose points where the surrogate variance is large. When exploiting, we should choose points where the surrogate mean is high.

It is highly desirable for our purposes to express $EI(\cdot)$ in a generalized form which controls the trade-off between global search and local optimization (exploration/exploitation). Schonlau [1997] suggests adding a non-negative integer parameter γ, such that $I(\mathbf{x}_{n+1}) = \max\{0, (\mu(\mathbf{x}_{n+1}) - f^{\max})^\gamma\}$, which results in an expected improvement of

$$EI_\gamma(\mathbf{x}_{n+1}) = s^\gamma(\mathbf{x}_{n+1}) \sum_{j=0}^{\gamma} (-1)^j \left(\frac{\gamma!}{j!(\gamma - j)!} \right) d^{\gamma - j} T_j,$$

where $T_j = -\phi(d) d^{j-1} + (j-1) T_{j-2}$, starting with $T_0 = \Phi(d)$ and $T_1 = -\phi(d)$. When $\gamma = 1$, (which degenerates to Jones' choice), emphasis is placed on trying to improve near $f^{\max}$, unless the observations strongly suggest improvement in areas of high variance. As γ is increased, areas of high model uncertainty will be favoured. While there is no obvious way to select γ for an arbitrary function, Sasena [2002] proposes an annealing-type schedule to allow global search to smoothly collapse to local improvement.

Lizotte [2008] suggests an alternative $\xi \geq 0$ parameter such that:

$$EI_\xi(\mathbf{x}) \quad = \quad \begin{cases} (\mu(\mathbf{x}) - (f^{\max} + \xi))\Phi(Z_\xi) + \sigma^2(\mathbf{x})\phi(Z_\xi) & \text{if } \sigma^2(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma^2(\mathbf{x}) = 0 \end{cases},$$

where

$$Z_\xi \quad = \quad \begin{cases} \frac{\mu(\mathbf{x}) - (f^{\max} + \xi)}{\sigma^s(\mathbf{x})} & \text{if } \sigma^2(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma^2(\mathbf{x}) = 0 \end{cases}.$$

Lizotte's experiments suggest that setting $\xi = \frac{\hat{\sigma}_f^2}{100}$ works well in almost all cases, and interestingly, setting a cooling schedule for ξ to encourage exploration early and exploitation later does *not* work well, contrary to intuition.

5 Bayesian Optimization for Preference Galleries

The model described above requires that each function evaluation have a scalar response. However, this is not always the case. In applications requiring human judgement, for instance, it is often easier to get preference data. We present here a Bayesian optimization application based on discrete choice for a "preference gallery" application, originally presented in [Brochu and de Freitas, 2006; Brochu *et al.*, 2007], though it could be easily extended to other discrete choice models.

Probability models for learning from discrete choices have a long history in psychology and econometrics [Thurstone, 1927; Mosteller, 1951; Stern, 1990; McFadden, 2001]. They have been studied extensively for use in rating chess players, and the Elo system [Élő, 1978] was adopted by the World Chess Federation FIDE to model the probability of one player beating another. It has since been adopted to many other two-player games such as Go and Scrabble, and, more recently, it has been generalized for online computer gaming [Herbrich and Graepel, 2006]. These methods differ from our work in that they are intended to predict the probability of a preference outcome over a finite set of possible pairs, whereas we work with infinite sets and are only incidentally interested in modelling outcomes. Glickman and Jensen [2005] use Bayesian optimal design for adaptively finding pairs for tournaments. Like our method, this work uses the results of previous comparisons to select the next pair to be evaluated, and can be considered a form of active learning. It differs from our method in that it adopts different utility models and, more importantly, in the fact that the number of items being compared is finite.

Parts of our method are based on [Chu and Ghahramani, 2005b], which presents a preference learning method using probit models and Gaussian processes. They use a Thurstone-Mosteller model, but with an innovative nonparametric model of the valuation function. [Chu and Ghahramani, 2005a] adds active learning to the model, though the method presented there differs from ours in that realizations are selected from a finite pool to maximize informativeness. Instead, we seek to maximize expected improvement over an infinite pool of items.

5.1 Design Galleries

Computer graphics is an important frontier of applied machine learning research [Hertzmann, 2003]. *Design galleries* [Marks *et al.*, 1997] is perhaps the best known assistance tool for animators. It is a browsing tool where a set of animations is displayed on a 2D layout using multi-dimensional scaling. It uses heuristics to find the set of input parameters to be used in the generation of the display. We depart from this heuristic treatment and instead present a principled probabilistic decision making approach to model the design process.

[Ledda *et al.*, 2005] used preference to conduct psychoperceptual experiments to evaluate tone mapping operators. Participants were presented with pairs of images and asked to indicate which they thought most closely resembled a reference scene. This approach is very similar in spirit to our system, though it uses a finite set of data and does not actively select pairs.

[Ngan *et al.*, 2006] have presented an interface for navigation in a perceptually uniform BRDF space based on a metric derived from user studies.

In this section, we detail the work we have done to date with active preference learning. While not the last word on the subject, this works well enough for us to have run some experiments (Section 5.3) and developed two applications employing these techniques: an interactive smoke simulation (Section 5.4) and a Bidirectional Radial Basis Function gallery tool (Section 5.5).

5.2 Active Preference Learning with Bayesian Optimization

By querying the user with a paired comparison, one can estimate statistics of the valuation function at the query point, but only at considerable expense. Thus, we wish to make sure that the samples we do draw will generate the maximum possible improvement.

Our method for achieving this goal iterates between the following steps:

1. **Present the user with a new pair and record the choice**: Augment the training set of paired choices with the new user data.
2. **Infer the valuation function**: Here we use a Thurstone-Mosteller model with Gaussian processes [Chu and Ghahramani, 2005b]. See Section 3.2.2 for details. Note that in this application. the valuation function is the objective of Bayesian optimization. We will use the terms interchangeably.
3. **Optimize the acquisition function of the valuation to obtain the next query point**: Here, we can simply use the $EI_\xi(\cdot)$ function.

5.3 Experiments

We have conducted a series of experiments to demonstrate the effectiveness of our methods.

In order to measure our method's effectiveness in finding the optimum of a function, we create a function f for which the optimum is known. At each time step, a query is generated in which two points $\mathbf{x}_1$ and $\mathbf{x}_2$ are selected according to the method described above, and the preference is found, where $f(\mathbf{x}_1) > f(\mathbf{x}_2) \Leftrightarrow \mathbf{x}_1 \succ \mathbf{x}_2$. To simulate noise in the preference relation, a noise term can be added to f. The learning machinery is then invoked to estimate the latent function.

The goal of our method is to successfully find the value of $\mathbf{x}$ that maximizes $f(\mathbf{x})$, so we measure the algorithm's performance based on how far its estimate of $\operatorname{argmax}_x f^*(\mathbf{x})$ is from the actual $\operatorname{argmax}_x f(\mathbf{x})$. At each time step we measure the error of the predictor as $\epsilon = |\operatorname{argmax}_x f^*(\mathbf{x}) - \operatorname{argmax}_x f(\mathbf{x})|$, that is, the distance from the point predicted to optimize $f^*(\mathbf{x})$ from the point that actually maximizes it. Note that by design, we do not penalize the algorithm for drawing samples from $\mathcal{X}$ that are far from argmax_x, or for predicting a latent function that differs from the true function.

5.3.1 Results

We measured the performance of our method on three functions – one-, two- and four-dimensional. By way of demonstration, Figures 4 and 5 show the actual functions and the typical predictions after a number of queries. The test

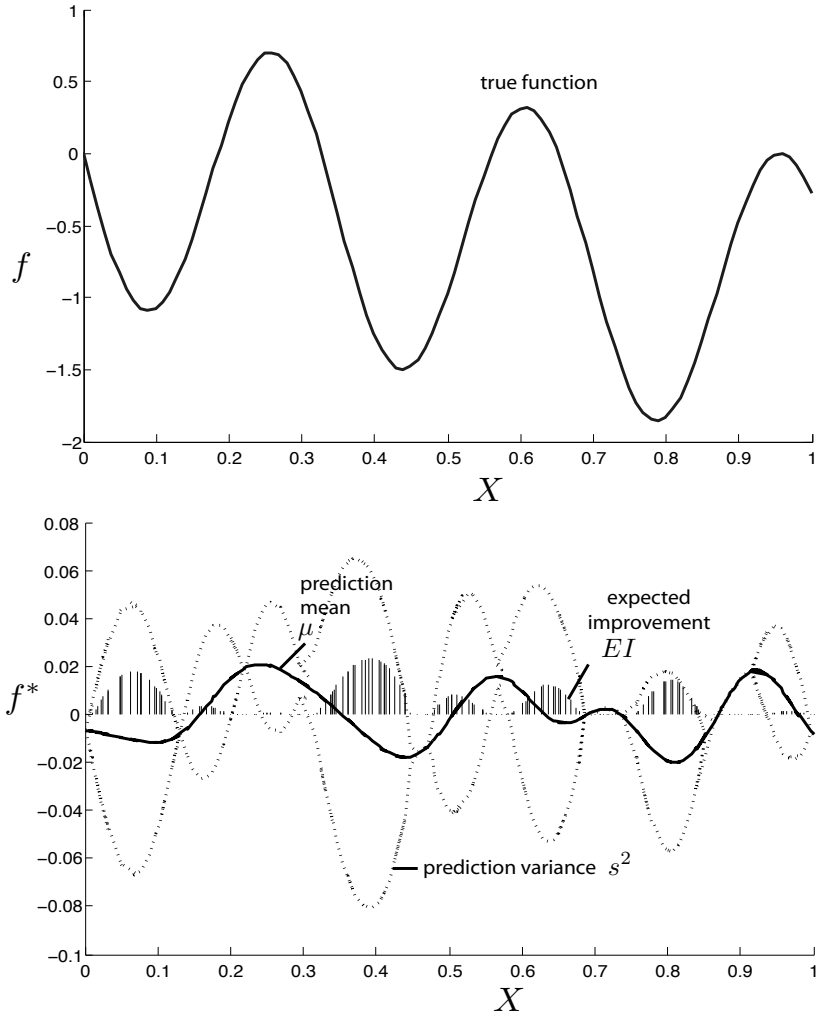


Figure 4: *The 1D test function (solid line), and the estimate of the function based on the results of a typical run of 9 preference queries. The predicted mean, $\mu(\mathbf{x})$, and variance $s^2(\mathbf{x})$ are used to compute the expected improvement function, $EI(\mathbf{x})$. Note that our method only fits the functions up to a scalar, and the emphasis is on finding the maximum, not fitting the entire function. This is the intent of our algorithm – we are not interested in predicting the response surface over the entire feasible domain, but rather in predicting accurately near the optimum.*

functions are defined as:

$$\begin{aligned} f_{1d} &= -\sin(x) - x/5 - \sin(18x)) \\ f_{2d} &= \max\{0, \sin(x_1) + x_1/3 + \sin(12x_1) \\ &\quad + \sin(x_2) + x_2/3 + \sin(12x_2) - 1\} \\ f_{4d} &= \sum_{i=1}^{4} \sin(x_i) + \overset{18}{x_i}/3 + \sin(12x_i) \end{aligned}$$

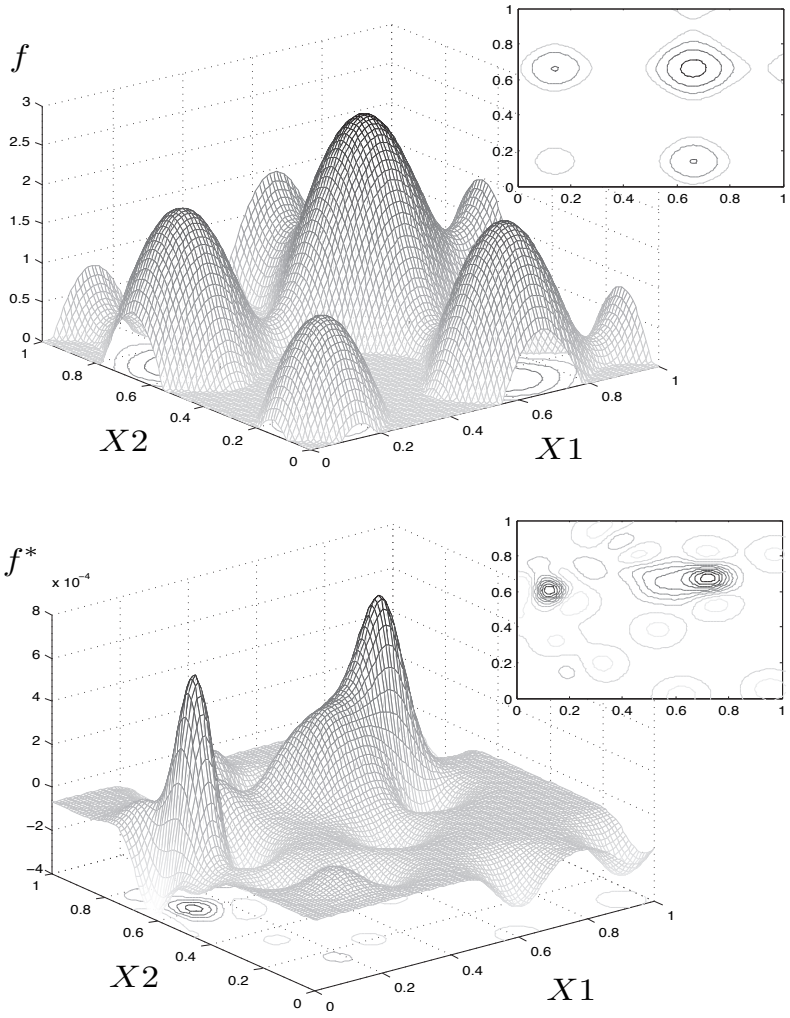


Figure 5: *The 2D test function (top), and the estimate of the function based on the results of a typical run of 12 preference queries (bottom). The true function has four local and one global maxima. As with the 1D function in Figure 4, the predictor identifies the region of the global maximum correctly and that of the local maxima less well.*

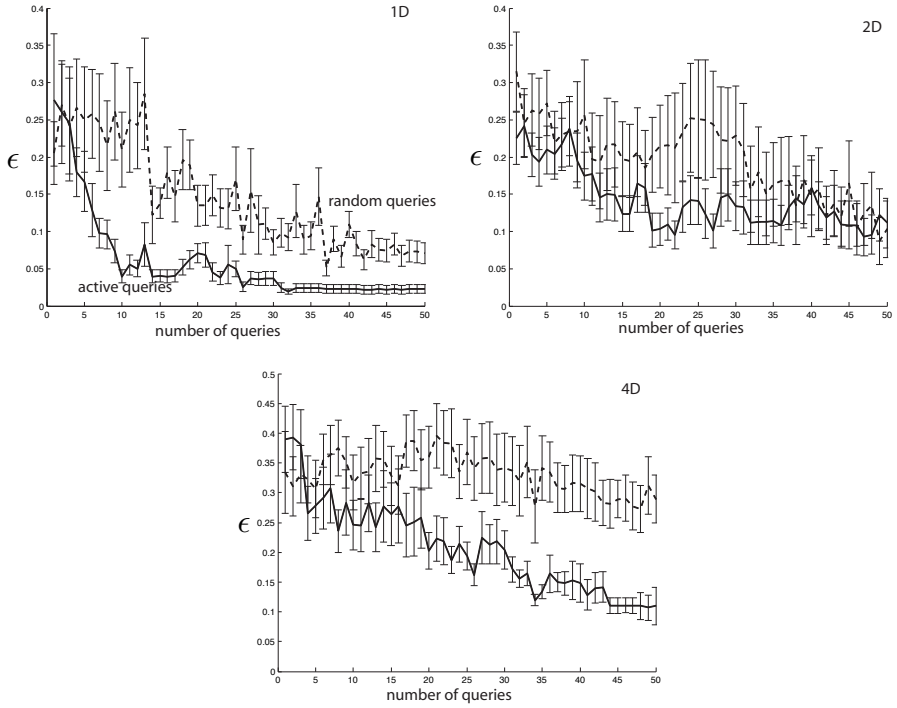


Figure 6: *The evolution of error for the estimate of the optimum on the one- and two-dimensional function shown in Figures 4 and 5, and a four-dimensional function. The error is defined as* $\epsilon = |\operatorname{argmax}_x f^*(x) - \operatorname{argmax}_x f(x)|$. *The plot shows the error rate against the number of queries. The solid line is our method; the dashed is a baseline comparison in which each query point is selected randomly. The performance is averaged over 20 runs, with the error bars showing the variance of* ϵ. *On the 1D case, our method typically finds a better estimate of the optimum by the tenth query than the random sampling does after 50, and does so quite consistently (as is shown by the low variance). The 2D case is somewhat less pronounced, but the high variance of the baseline indicates it is frequently identifying a local maxima as the global maximum. On the 4D case, random sampling barely improves even after 50 queries, whereas our method quickly finds a good estimate and steadily improves.*

all defined over the range $[0, 1]^d$. We selected these equations because they seem both general and difficult enough that we can safely assume that if our method works well on them, it should work on a large class of real-world problems — they have multiple local maxima to get trapped in and varying landscapes and dimensionality. Unfortunately, there has been little work in the psychoperception literature to indicate what a good test function would be for our problem, so we have had to rely to an extent on our intuition to develop suitable test cases.

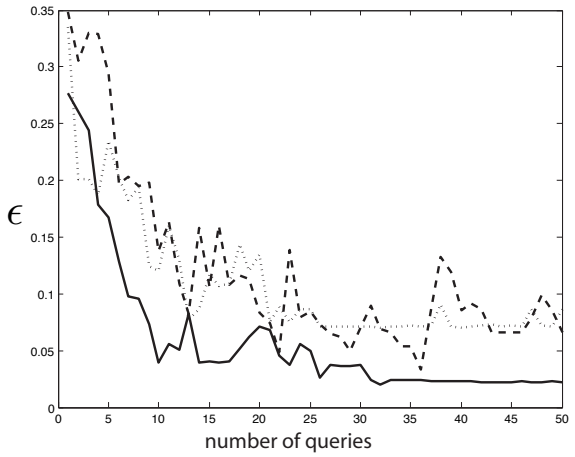


Figure 7: *Three methods of selecting queries for the one-dimensional test function (other functions not shown for space reasons, but results are similar). This shows the empirical results of comparing the point of maximum expected improvement,* $\mathbf{x}_{best}$*, to: the previously preferred point,* $\mathbf{r}_{k-1}$ *(solid); the second-highest local maximum EI (dashed); and a randomly selected point in* $\mathbf{r}_{1:k-1}$*. Result is the average error evolution over 20 runs of 50 queries each. The* $\{\mathbf{x}_{best}, \mathbf{r}_{k-1}\}$ *comparison is the clear winner, and is used in our other experiments.*

The results of the experiments are shown in Figure 6. In all cases, we simulate 50 queries using our method. As a baseline, we compare against 50 random queries to see how long uninformed sampling takes to model the space. We repeated each experiment 20 times and measured the mean error and the variance of the error. A high variance can often be an indicator that the estimator is frequently misidentifying a local maximum as the global maximum.

For 1D and 2D problems, both random sampling and our method eventually come close to the global maximum. However, active choice selection finds the maximum much more rapidly – typically, the error of our method is as good after 10 queries as the random sampling is after 50. For even 4D, though, random sampling is simply unable to find a good approximation in the first 50 queries, while our method quickly and steadily improves with each query.

5.3.2 Choice Selection

While intuitively, it seems reasonable to include the point of maximum expected improvement, $\mathbf{x}_{best}$ as one of the choices, it is not necessarily obvious what that point should be compared to. Intuitively, we felt that at each iteration k, comparing to $\mathbf{r}_{k-1}$ would be the most likely to guarantee that the preferred choice would, in fact, be the best known point. However, other options are available, and we elected to compare them empirically.

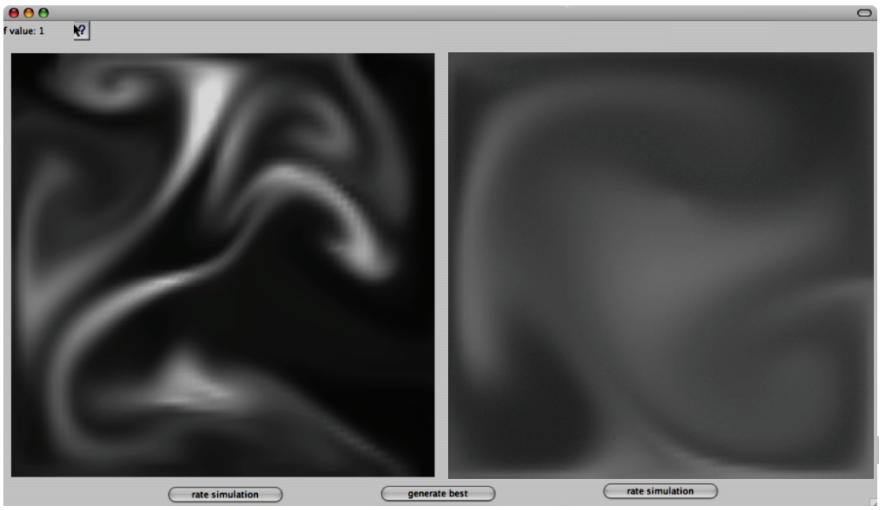


Figure 8: *An example of our active-learning-based smoke simulation comparison tool. The simulation engine is interactive and real-time, but controlling the simulation environment requires the setting of five interacting parameters. Our system allows a user with no prior experience to find the desired simulation parameters by generating simulations for the user to compare. At any point, the user can also ask the system to predict the current estimated best simulation parameters based on his or her feedback.*

In the first alternative scenario, instead of using the preferred choice of the previous query, we randomly select a point from all the previous winners, $\mathbf{r}_{1:k-1}$. This was intended to permit the re-examination of choices were known to be good, but that might have been otherwise overlooked. The second alternative scenario was to compare the point of maximum expected improvement to the second-highest local maximum, $\mathbf{x}_{2ndbest}$. The intent of this is to more quickly explore regions of high expected improvement, since often both points will then be in areas that were previously unexplored.

A comparison of the methods for the 1D case is shown in Figure 7. Plots for the other test functions are omitted for reasons of space, but are similar. The clear winner is the choice $\{\mathbf{x}_{best}, \mathbf{r}_{k-1}\}$. We speculate that the poor results of the comparison with the second-highest maximum, $\{\mathbf{x}_{best}, \mathbf{x}_{2ndbest}\}$ are due to the fact that the method often adds preference relations that often cannot be accurately compared to previously-known preference relations, as they may both be in regions of high uncertainty. More investigation will be required to confirm or confute this.

5.4 Active Preferences for Smoke Simulation

The initial motivation for this work was an animation tool based on Jos Stam's smoke simulation [Stam, 2003]. This is a 2D simulation, which is not necessarily intended to be physically accurate, but rather to generate smoke effects that look "realistic". It uses the familiar Navier-Stokes equations and a grid of samples on a density field. At each time step, the density field is updated by applying a velocity field (representing air flow or other interference, also sampled on a grid), uniform diffusion, and additional sources of density (that is, smoke added to the system from a source point).

It uses the familiar Navier-Stokes equations.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \\ \frac{\partial \rho}{\partial t} &= -(\mathbf{u} \cdot \nabla) + \kappa \nabla^2 \rho + S \end{aligned}$$

The simulation has a vector $\mathbf{x}$ of five parameters, all of which take positive real values: *viscosity*, *diffusion*, *time step*, *source* (the amount of smoke generated in the simulation when the user clicks the simulation window) and *force* (the amount of disruption the user's mouse click creates in the simulation). Clearly, some of these are dependent (*force* and *time step*, for example). The system is extremely robust, and will not blow up even when given bad values, but large parts of the parameter space generate simulations that look disappointing, at best. Furthermore, this real-time simulation can generate a variety of legitimate smoke, from wispy curlicues like cigarette smoke to thick, viscous smoke resembling factory exhaust.

Our simulation-rating environment is a GUI application shown in Figure 8. The user-feedback portion of our application allows the user to interact with two simulation environments and indicate preference using whatever desired valuation measure they have in mind, typically a specific smoke effect. By clicking mouse buttons, the user can add and move smoke within either simulation window. A window button allows the user to indicate preference, at which point our algorithm is run again and selects two more simulations to show the user. At any time, the user can click a button in the GUI to get the "best" simulation, which is computed by running DIRECT to find an $\mathbf{x}$ (set of parameters) that maximizes the valuation rather than $EI(\cdot)$.

If this is not the final result the user sought, it can be rated as usual, and another iteration of our algorithm is performed.

It is difficult to provide an objective evaluation of an animation design tool on paper. In the future we intend to embark on extensive user studies to extend the informal user studies we have already performed in developing the application.

5.5 Active Preferences for Material Design

Properly modeling the appearance of a material is a necessary component of realistic image synthesis. The appearance of a material is formalized by the

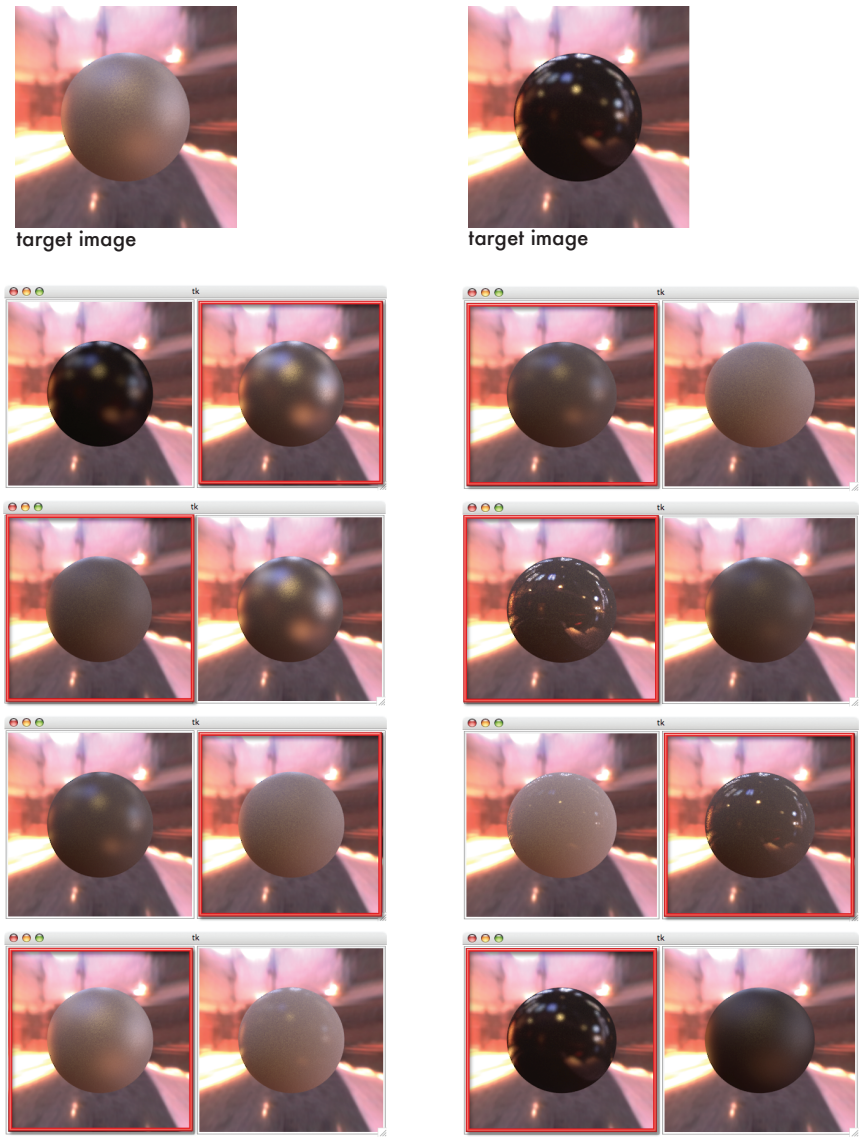


Figure 9: *Two example sequences of user selections (highlighted in red) of BRDFs in our preference gallery. The user is supplied with the indicated target image and the task of finding the image using the preference interface. Top-pairs: Initial choices provided by the gallery. Center-pairs: Intermediate choices. Bottom-pairs: Final choices provided by the gallery before the user converges to the target appearance. In most runs, the user was able to find the image with 4 or 5 preferences.*

notion of the Bidirectional Reflectance Distribution Function (BRDF). In computer graphics, BRDFs are most often specified using various analytical models. Analytical models that are of interest to realistic image synthesis are the ones that observe the physical laws of reciprocity and energy conservation while typically also exhibiting shadowing, masking and Fresnel reflectance phenomenon. Realistic models are therefore fairly complex with many parameters that need to be adjusted by the designer for the proper material appearance. Unfortunately these parameters can interact in non-intuitive ways, and small adjustments to certain settings may result in non-uniform changes in the appearance. This can make the material design process quite difficult for the artist end user, who is not expected to be an expert in the field. To alleviate this problem, Ngan et al. [2006] presented an interface for navigation in a perceptually uniform BRDF space based on a metric derived from user studies. However, this is still somewhat constraining as the user has to develop an understanding of the various aspects of material appearance such as varying degrees of diffuseness, glossiness, specularity, Fresnel effects and/or anisotropy in order to navigate such an interface. An artist often knows the look that she desires for a particular application without necessarily being interested in understanding the various subtleties of reflection. We attempt to deal with this using a "preference gallery" approach, in which users are simply required to view two or more images rendered with different material properties and indicate which they prefer, in an iterative process.

To maximize the valuation, we use an implementation of the model described in Section 5.2. Maxima may be points of high variance or high predicted valuation, or both. In practice, the first few examples will be points of high variance, since little of the space is explored (that is, the model of user valuation is very uncertain). Later samples will tend to be in regions of high valuation, as a model of the user's interest is learned. Note that we are *not* trying to learn the entire valuation function, which would take many more queries – we seek only to maximize the user's valuation, which involves accurate modelling only in the areas of high valuation.

We use our active preference learning model on an example gallery application for helping users find a BRDF. For the purposes of this example, we limit ourselves to isotropic materials and ignore wavelength dependent effects in reflection. The gallery uses the Ashikhmin-Shirley Phong model for the BRDFs and the Grace Cathedral HDR environment illumination. Our gallery demonstration presents the user with two BRDF images at a time. We start with four predetermined queries to "seed" the parameter space, and after that use the learned model to select gallery images. The GP model is updated after each preference is indicated. We use parameters of real measured materials from the MERL database for seeding the parameter space, but can draw arbitrary parameters after that.

To evaluate the performance of our application, we have run a simple user study in which the generated images are restricted to a random subset of 38 images. The user is given the task of finding a single randomly-selected image from that set by indicating preferences. Figure 9 shows a typical user run, where

we ask the user to use the preference gallery to find a provided target image. At each step, the user need only to indicate the image they think looks most like the target. Using image pairs, it takes an average of 4 to 5 selections for the user to arrive at the target material appearance, depending on the material. Random selection requires approximately twice as many trials to find an acceptable (to the user) match. Obviously, this study is too simplistic to say anything definite, but it is encouraging, and we intend to use this framework to conduct more sophisticated tests.

6 Bayesian Optimization for Hierarchical Control

In general, problem solving and planning becomes easier when it is broken down into subparts. Variants of functional hierarchies appear consistently in video game AI solutions, from behaviour trees, to hierarchically decomposed agents (teams vs. players), implemented by a multitude of customized hierarchical state machines. The benefits are due to isolating complex decision logic to fairly independent functional units (tasks). The standard game AI development process consists of the programmer implementing a large number of behaviours (in as many ways as there are published video games), and hooking them up to a more manageable number of tuneable parameters. We present a class of algorithms that attempt to bridge the gap between game development, and general reinforcement learning. They reduce the amount of hand-tuning traditionally encountered during game development, while still maintaining the full flexibility of manually hard-coding a policy when necessary.

The Hierarchical Reinforcement Learning [Barto and Mahadevan, 2003] field models repeated decision making by structuring the policy into tasks (actions) composed of subtasks that extend through time (temporal abstraction) and are specific to a subset of the total world state space (state abstraction). Many algorithms have recently been developed, and are described further in Section 6.1.

The exploration policies typically employed in HRL research tend to be slow in practice, even after the benefits of state abstraction and reward shaping. We demonstrate an integration of the MAXQ hierarchical task learner with Bayesian active exploration that significantly speeds up the learning process, applied to hybrid discrete and continuous state and action spaces. Section 6.2 describes an extended Taxi domain, running under The Open Racing Car Simulator [Wymann *et al.*, 2009], a 3D game engine that implements complex vehicle dynamics complete with manual and automatic transmission, engine, clutch, tire, suspension and aerodynamic models.

6.1 Hierarchical Reinforcement Learning

Manually coding hierarchical policies is the mainstay of video game AI development. The requirements for automated HRL to be a viable solution are it must be easy to customize task-specific implementations, state abstractions, reward

models, termination criteria and it must support continuous state and action spaces. Out of the solutions investigated, MAXQ [Dietterich, 2000] met all our requirements, and was the easiest to understand and get positive results quickly. The other solutions investigated include HAR and RAR [Ghavamzadeh, 2005] which extend MAXQ to the case of average rewards (rather than discounted rewards). The implementation of RAR is mostly the same as MAXQ, and in our experiments gave the same results. Hierarchies of Abstract Machines (HAM) [Parr, 1998] and ALisp [Andre, 2003] are an exciting new development that has been recently applied to a Real-Time-Strategy (RTS) game [Marthi *et al.*, 2005]. ALisp introduces programmable reinforcement learning policies that allows the programmer to specify choice points for the algorithm to optimize. Although the formulation is very nice and would match game AI development processes, the underlying solver based on HAMs flattens the task hierarchy by including the program's memory and call-stack into a new joint-state space, and solves this new MDP instead. It is less clear how to extend and implement per-task customized learning with this formulation. Even if this difficulty is surmounted, as evidenced by the last line in the concluding remarks of [Marthi *et al.*, 2005], there is an imperative need for designing faster algorithms in HRL. This paper aims to address this need.

In our solution, we still require a programmer or designer to specify the task hierarchy. In most cases breaking a plan into sub-plans is much easier than coding the decision logic. With the policy space constrained by the task hierarchy, termination and state abstraction functions, the rate of learning is greatly improved, and the amount of memory required to store the solution reduces. The benefits of HRL are very dependant however on the quality of these specifications, and requires the higher-level reasoning of a programmer or designer. An automatic solution to this problem would be an agent that can learn how to program, and anything less than that will have limited applicability.

We can use Bayesian optimization to learn the relevant aspects of value functions by focusing on the most relevant parts of the parameter space. In the work on this section, we use refer to the objective as the *value* function, to be consistent with the HRL literature.

6.1.1 Semi-MDPs

Each task in an HRL hierarchy is a semi-Markov Decision Process [Sutton *et al.*, 1999], that models repeated decision making in a stochastic environment, where the actions can take more than one timestep. Formally, an SMDP is defined as a tuple: $\{S, A, P(s', N|s, a), R(s, a)\}$ where S is the set of state variables, A is a set of actions, $P(s', N|s, a)$ is the transition probability of arriving to state s' in N timesteps after taking action a in s, and $R(s, a)$ is the reward received. The solution of this process is a policy $\pi^*(s) \in A$, that selects the action with the highest expected discounted reward in each state. The function $V^*(s)$ is the value of state s when following the optimal policy. Equivalently, the $Q^*(s, a)$ function stores the value of taking action a in state s and following the optimal policy thereafter. These quantities follow the classical Bellman recursions:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s',N} P(s',N|s,a)\gamma^N V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s',N} P(s',N|s,a)\gamma^N V^*(s') \tag{6}$$

6.1.2 Hierarchical Value Function Decomposition

A task i in MAXQ [Dietterich, 2000] is defined as a tuple: $\{A_i, T_i(s), Z_i(s), \pi_i(s)\}$ where s is the current world state, A_i is a set of subtasks, $T_i(s) \in \{true, false\}$ is a termination predicate, $Z_i(s)$ is a state abstraction function that returns a subset of the state relevant to the current subtask, and $\pi_i(s) \in A_i$ is the policy learned by the agent (or used to explore during learning). Each task is effectively a separate, decomposed SMDP that has allowed us to integrate active learning for discrete map navigation with continuous low-level vehicle control. This is accomplished by decomposing the Q function into two parts:

$$\begin{aligned} a &= \pi_i(s) \\ Q^\pi(i,s,a) &= V^\pi(a,s) + C^\pi(i,s,a) \\ C^\pi(i,s,a) &= \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi_i(s')) \\ V^\pi(i,s) &= \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if primitive} \end{cases} \end{aligned} \tag{7}$$

Here, γ is the discount factor, i is the current task, and a is a child action given that we are following policy π_i. The Q function is decomposed into two parts: the value of V^π being the expected one step reward, plus C^π which is the expected completion reward for i after a completes. V is defined recursively, as the expected value of its child actions, or the expected reward itself if i is a primitive (atomic) action. The MAXQ learning routine is a simple modification of the typical Q-learning algorithm. In task i, we execute subtask a, observe the new state s' and reward r. If a is primitive, we update $V(s,a)$, otherwise we update $C(i,s,a)$, with learning rate $\alpha \in (0,1)$:

$$\begin{aligned} V(a,s) &= (1-\alpha) \times V(a,s) + \alpha \times r \\ C(i,s,a) &= (1-\alpha) \times C(i,s,a) + \alpha \times \max_{a'} Q(i,s',a') \end{aligned} \tag{8}$$

An important consideration in HRL is whether the policy calculated is hierarchically or recursively optimal. Recursive optimality, satisfied by MAXQ and RAR, means that each subtask is locally optimal, given the optimal policies of the descendants. This may result in a suboptimal overall policy because the effects of tasks executed outside of the current task's scope are ignored. For example if there are two exits from a room, a recursively optimal policy would

pick the closest exit, regardless of the final destination. A hierarchically optimal policy (computed by the HAR [Ghavamzadeh, 2005] and HAM [Andre, 2003] three-part value decompositions) would pick the exit to minimize total travelling time, given the destination. A recursively optimal learning algorithm however generalizes subtasks easier since they only depend on the local state, ignoring what would happen after the current task finishes. So both types of optimality are of value in different degrees for different cases. The MAXQ formulation gives a programmer or designer the ability to selectively enable hierarchical optimality by including the relevant state features as parameters to a task. However, it may be difficult to identify the relevant features, as they would be highly application specific.

6.2 The Vancouver Taxi Domain

Our domain is a city map roughly based on a portion of downtown Vancouver, British Columbia, illustrated in Figure 10. The data structure is a topological map (a set of intersection nodes and adjacency matrix) with 61 nodes and 22 possible passenger pickup and drop-off locations. The total navigable area of the map is roughly 28 kilometers.

The state model includes both discrete variables used in the top layers of the task hierarchy, as well as continuous variables used by the *Follow* task that tracks a trajectory, and are described in Table 2. The original taxi domain [Dietterich, 2000] is a 5x5 grid, with 4 possible pickup and dropoff destinations, and 6 actions (pickup, dropoff, and navigating North, South, East, West).

Table 1 makes a rough comparison between the size of our extended application and the original taxi domain. Ignoring the continuous trajectory states (including the *Stopped* flag) and assuming the taxi hops from one intersection to an adjacent one in a single timestep results in a fully discrete problem. A flat learning solution scales poorly, not only in terms of world samples required, but also in the size of the computed policy (if represented in a discrete table). The extended task hierarchy illustrated in Figure 11 requires just a little bit more memory than the small 5x5 taxi domain.

Table 1: Comparing Domain Size

Domain	Size of final policy
5x5 Taxi Flat	$\sim 12,200$ *bytes*
Vancity Flat	$\sim 1,417,152$ *bytes*
Vancity Hierarchical	$\sim 18,668$ *bytes*

6.2.1 State Abstraction, Termination and Rewards

Figure 11 compares the original task hierarchy, with our extended version that includes continuous trajectory following and a hard-coded *Park* task. The state

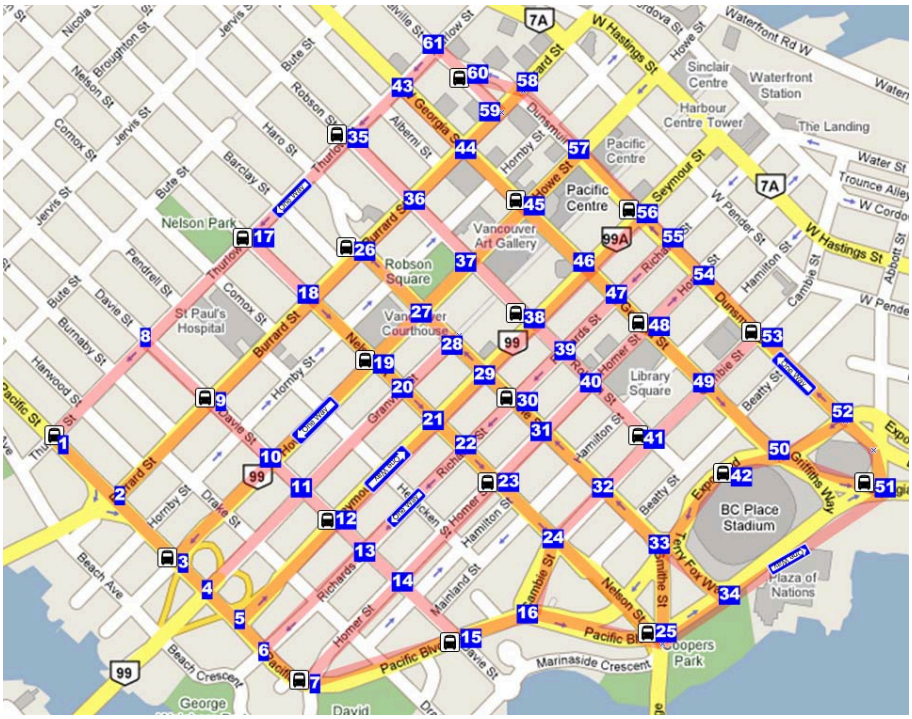


Figure 10: ***City Experiment*** *uses a simplified map (orange overlay) roughly based on downtown Vancouver, and used by the TORCS simulator. Each waypoint is labeled, and pickup and dropoff locations are marked by the Taxi icons. One way streets are accounted for in the waypoint adjacency matrix. Source image care of Google Maps.*

abstraction function filters out irrelevant states while computing the hash key for looking up and updating values of $V(s,a)$ and $C(i,s,a)$, where s is the current state, i is the current task, and a is the child task. The *Follow* task has been previously trained with the Active Policy optimizer from section 6.3.1 and the policy parameters fixed before learning the higher level tasks. Algorithms RAR and MAXQ are applied to all the tasks above and including *Navigate*, which also uses the Active Path learning algorithm from section 6.3.2. Here is a summary of each task, including its reward model, termination predicate T_i, and state abstraction function Z_i:

Root - this task selects between *Get* and *Put* to pickup and deliver the passenger. It requires no learning because the termination criteria of the subtasks fully determine when they should be invoked. $T_{Root} = (PassLock = PassDest)$ and $Z_{Root} = \{\}$.

Get - getting the passenger involves navigating through the city, parking the car and picking up the passenger. In this task, the *LegalLoad* state is true when the taxi is at the passenger's location. Receives a reward of 750 when the passenger is picked up, $T_{Get} = ((PassLoc = 0)\ or\ (PassLoc = PassDest))$,

Table 2: States and Task Parameters

Name	Range/Units	Description
$TaxiLoc$	{0,1,..61}	current taxi waypoint #, or 0 if in transit between waypoints
$PassLoc$	{0,1,..22}	passenger waypoint #, or 0 if in taxi
$PassDest$	{1,2,..22}	passenger destination waypoint #
$LegalLoad$	$\{true, false\}$	true if taxi is empty and at passenger, or loaded and at target
$Stopped$	$\{true, false\}$	indicates whether the taxi is at a complete stop
T	{1,2,..22}	passenger location or destination parameter passed into *Navigate*
WP	{1,2,..22}	waypoint parameter adjacent to *TaxiLoc* passed to *Follow*
Y_{err}	meters	lateral error between desired point on the trajectory and vehicle
V_y	meters/second	lateral velocity (to detect drift)
V_{err}	meters/second	error between desired and real speed
Ω_{err}	radians	error between trajectory angle and vehicle yaw

and $Z_{Get} = \{\}$.

Put - similar to *Get*, also receives reward of 750 when passenger is successfully delivered. The passenger destination *PassDest* is passed to the *Navigate* task. The abstracted *LegalLoad* state is true when the taxi is at the passenger's destination location. $T_{Put} = ((PassLoc > 0)\ or\ (PassLoc = PassDest))$ and $Z_{Put} = \{\}$.

Pickup - this is a primitive action, with a reward of 0 if successful, and -2500 if a pickup is invalid (if the taxi is not stopped, or if *LegalLoad* is false). $Z_{Pickup} = \{LegalLoad, Stopped\}$.

Dropoff - this is a primitive action, with a reward of 1500 if successful, and -2500 if a dropoff is invalid. $Z_{Dropoff} = \{LegalLoad, Stopped\}$.

Navigate - this task learns the sequence of intersections from the current *TaxiLoc* to a target destination T. By parameterizing the value function of this task, we can apply Active Path learning as described in Section 6.3.2. $T_{Navigate} = (TaxiLoc = T)$ and $Z_{Navigate} = \{T, TaxiLoc\}$.

Follow - this is the previously trained continuous trajectory following task that takes as input an adjacent waypoint *WP*, and generates continuous steering and throttle values to follow the straight-line trajectory from *TaxiLoc* to *WP*. $T_{Follow} = (TaxiLoc = WP)$ and $Z_{Follow} = \{WP, \Omega_{err}, V_{err}, Y_{err}, V_y\}$.

Park - this is a hard-coded task which simply puts on the brakes ($steer = 0$, $throttle = -1$).

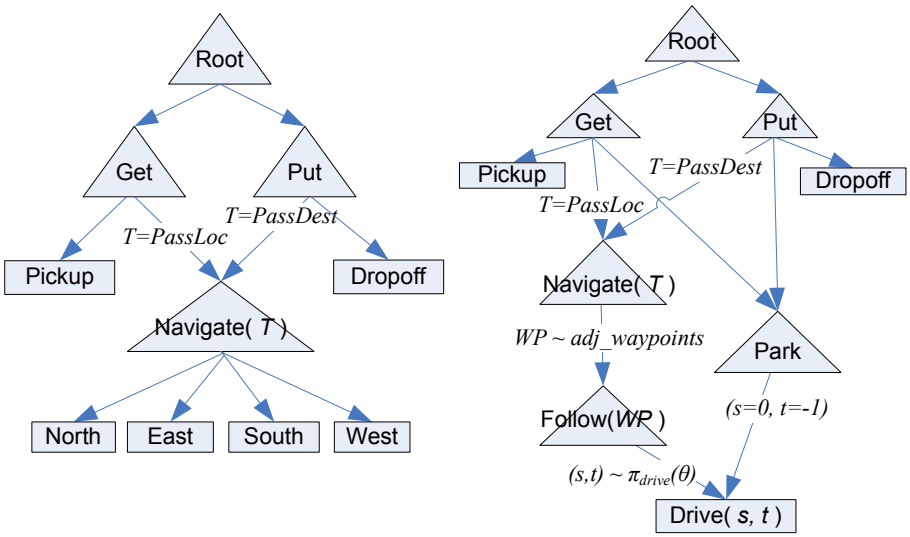


Figure 11: ***Task Hierarchies.*** Each composite task is a separate SMDP whose policy is optimal given the optimal policies of its subtasks (recursive optimality). Triangles are composite tasks, and rectangle are primitive actions. The hierarchy on the right simplifies learning by reusing policies for navigating form waypoint to waypoint, and the Navigation task only needs to learn the sequence of waypoints to get to the destination. For the continuous case, the discrete actions N/S/E/W are replaced by one continuous *Drive(steer, throttle)* task, with driving parameters generated by the parameterized policy contained in the *Follow* task.

Drive - this performs one timestep of the physics simulation, with the given steer and throttle inputs. The default reward per timestep of driving is -0.75.

6.3 Bayesian Optimization for Hierarchical Policies

The objective of Bayesian optimization is to learn properties of the value function or policy with as few samples as possible. In direct policy search, where this idea has been explored previously [Martinez-Cantin *et al.*, 2007], the evaluation of the expected returns using Monte Carlo simulations is very costly. One, therefore, needs to find a peak of this function with as few policy iterations as possible. As shown here, the same problem arises when we want to learn an approximation of the value function only over the relevant regions of the state space. Bayesian optimization provides an exploration-exploitation mechanism for finding these relevant regions and fitting the value function where needed.

When carrying out direct policy search [Ng and Jordan, 2000], the Bayesian optimization approach has several advantages over the policy gradients method [Baxter *et al.*, 2001]: it is derivative free, it is less prone to be caught in the first local minimum, and it is explicitly designed to minimize the number of

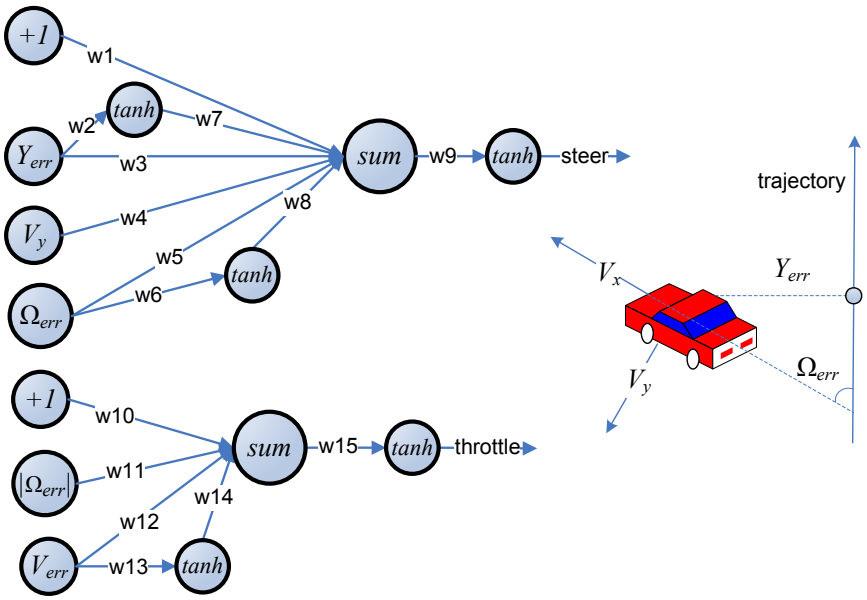


Figure 12: ***Trajectory-following policy:*** *this parameterized policy, inspired by Ng* et al *[2003] minimizes the error between the vehicle's heading and velocity while following a trajectory. The positional errors X_{err} and Y_{err} are in trajectory coordinates, Ω_{err} refers to the difference between the current heading and the trajectory tangent, and V_{err} is the difference between the real and desired velocities.*

expensive value function evaluations.

6.3.1 Active Policy Optimization

The lowest level *Drive* task uses the parameterized function illustrated in Figure 12 to generate continuous steer and throttle values, within the range of -1 to 1. The $|\theta| = 15$ parameters (weights) are trained using the Bayesian active policy learning Algorithm 1. We first generate and evaluate a set of 30 Latin hypercube samples of θ and store them and corresponding values vector V in the data matrix $\mathcal{D}$. The value of a trajectory is the negative accumulated error between the car's position and velocity, and the desired position and velocity. The policy evaluation consists of averaging 10 episodes along the same trajectory but with different, evenly spaced starting angles, where the car needs to accelerate from rest, go to the first waypoint, perform a u-turn, and arrive back to the starting location. In a noisier environment, more samples would be necessary to properly evaluate a policy. The TORCS simulator is deterministic, and a small amount of noise arises from un-modelled tire slipping and random bumpiness of the road. The 10 different starting angles were sufficient for evaluating a policy in our experiments. Subsequently, we perform the iteration described in Algorithm 1 to search for the best instantiation of the parameters.

6.3.2 Active Value Function Learning

The *Navigate* task learns path finding from any intersection in the topological map to any of the destinations. Although this task operates on a discrete set of waypoints, the underlying map coordinates are continuous, and we can again apply active exploration with GPs.

Unlike the previous algorithm that searches for a set of optimal parameters, Algorithm 2 learns the value function at a finite set of states, by actively generating exploratory actions; it is designed to fit within a MAXQ task hierarchy. The 4-dimensional value function $V(\theta)$ in this case is parameterized by two 2D map coordinates $\theta = \{x_C, y_C, x_T, y_T\}$, and stores the sum of discounted rewards while travelling from the current intersection $|C| = 61$ to the target $|T| = 22$. The sampled instances of $|\theta| = 1342$ and corresponding $V(\theta)$ vector are stored in the data matrix $\mathcal{D}$; it is initialized with $V(x_T, y_T, x_T, y_T) = 0$ for all target destinations T, which enables the GP to create a useful response surface without actually having observed anything yet.

In the $\epsilon-$greedy experiments, a random intersection is chosen with chance 0.1, and the greedy one with chance 0.9. For the active exploration case, we fit a GP over the data matrix $\mathcal{D}$, and pick the adjacent intersection that maximizes the expected improvement function. We parameterize this function with an annealing parameter that decays over time such that initially we place more importance on exploring.

The true value will not be known until the *Navigate* task reaches its destination and terminates, but we still need to mark visited intersections to avoid indefinite looping. Lines 23-26 compute an estimated value for $V(s)$ by summing the immediate discounted reward of executing $Follow(WP, s)$ with the discounted, previously recorded value of the new state $V(s')$, and a heuristic penalty factor to avoid looping. Once we reach the destination of this task, $Target_i$, we have the necessary information to propagate the discounted reward to all the intersections along the trajectory, in lines 15-21.

6.4 Simulations

The nature of the domain requires that we run policy optimization first to train the *Follow* task. This is reasonable, since the agent cannot be expected to learn map navigation before learning to drive the car. Figure 13 compares the results of three different values for the GP kernel k, when running the active policy optimization algorithm from section 6.3.1. The desired velocity is $60km/hr$, a timestep lasts 0.25 seconds, and the trajectory reward $R = -\sum_t \left[1 \times \tilde{Y}_{err}^2 + 0.8 \times \tilde{V}_{err}^2 + 1 \times \tilde{\Omega}_{err}^2 + 0.7 \times \tilde{\mathbf{a}}'\tilde{\mathbf{a}}\right]$ is the negative weighted sum of normalized squared error values between the vehicle and the desired trajectory, including $\mathbf{a} = [\mathbf{steer}, \mathbf{throttle}]$ to penalize for abrupt actions. After ~ 50 more parameter samples (after the initial 30 random samples), the learner has already found a useable policy.

Subsequently, the best parameters are fixed inside the *Follow* task, and we run the full hierarchical learners, with results in Figure 14. We averaged the re-

Algorithm 2 Active Path Learning with GPs

function $NavigateTaskLearner(Navigate\ i, State\ s)$

let $trajectory$=() - list of all states visited in i
let $intersections$=() - intersection states visited in i
let $visits = 0$ - # of visits at an intersection in i

while $Terminated_i(s)$ is false **do**
  choose adjacent intersection WP using ϵ-greedy or Active exploration.
  let $childSeq = Follow(WP, s)$
  append $childSeq$ onto the front of $trajectory$
  observe result state s'

  N = length($childSeq$)
  $R = \sum_{j=1}^{N} \gamma^{N-j} \times r_j$ be the total discounted reward received from s to s'

  $V'_s = V(TaxiLoc_{s'}, Target_i)$ { guaranteed <= 0}
  $V_s = V(TaxiLoc_s, Target_i)$ { guaranteed <= 0}
  if $Terminated_i(s')$ is true **then**
    $V_s \leftarrow (1 - \alpha) \times V_s + \alpha \times R$
    for all $j = 1$ to length(intersections) **do**
      $\{s', N', R'\}$ = intersections(j)
      $R \leftarrow R' + \gamma^{N'} \times R$
      $V'_s \leftarrow V(TaxiLoc_{s'}, Target_i)$
      $V'_s \leftarrow (1 - \alpha) \times V'_s + \alpha \times R$
    end for

  else
    append $\{s, N, R\}$ onto the front of $intersections$
    $visits(TaxiLoc_s) \leftarrow visits(TaxiLoc_s) + 1$
    $penalty \leftarrow V_s \times visits(TaxiLoc_s)$ {prevent loops}
    $V_s \leftarrow (1 - \alpha) \times V_s + \alpha(penalty + R + \gamma^N \times V'_s)$
  end if

  $s = s'$
end while

return $trajectory$

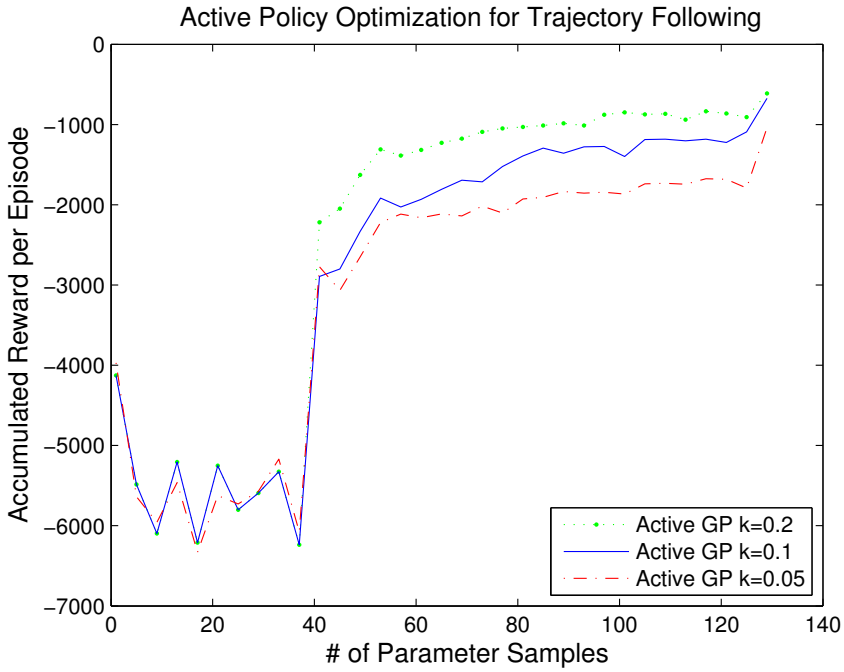


Figure 13: ***Active Policy Optimizer:*** *searching for the* 15 *policy parameters, and comparing different values for the GP kernel size* k*. We used the Expected Improvement function 5, and the three experiments are initialized with the the same set of* 30 *Latin hypercube samples. A total of* 20 *experimental runs were averaged for this plot.*

sults from 10 runs of RAR, MAXQ, and the value learning Algorithm 2 applied only to the *Navigate* task (with the rest of the hierarchy using MAXQ). All the experiments use the hierarchical task model presented in Section 6.2.1. Each reward timestep lasts 0.3 seconds, so the fastest learner, V_{TM} GP with $\epsilon = 0.2$ drove for ~ 4 hours real-time at ~ 60 km/hr before finding a good approximation of the $V_{Navigate}$ value function. Refer to Figure 15 for an intuition of how fitting the GP over the samples values transfers observations to adjacent areas of the state space. This effect is controlled through the GP kernel parameter k. While the application is specific to navigating a topological map, the algorithm is general and can be applied to any continuous state spaces of reasonable dimensionality.

7 Alternative Priors: Random Forests for Bayesian Optimization

As we have seen, Gaussian processes can be put to excellent effect in Bayesian optimization. However, there is nothing that *requires* us to use GPs as the

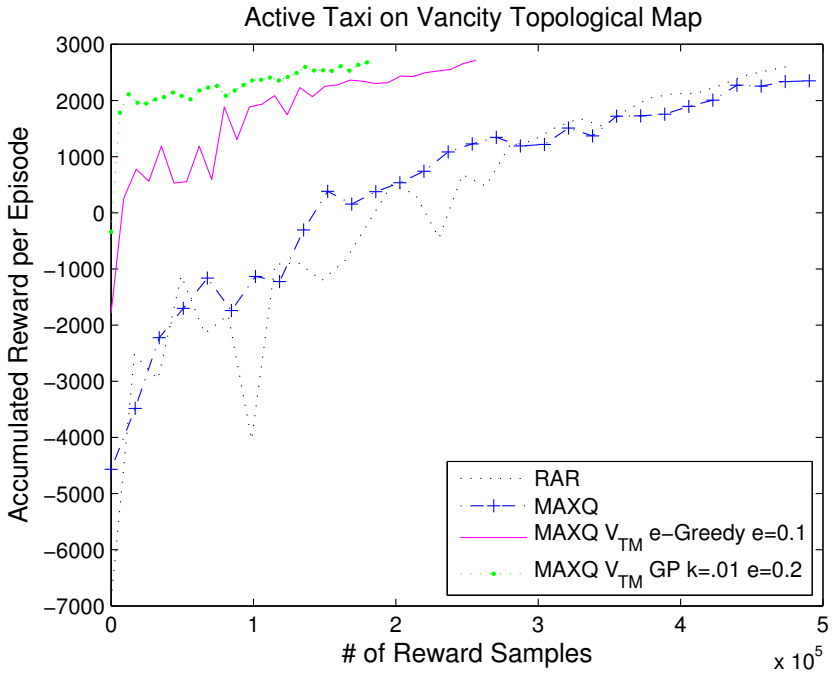
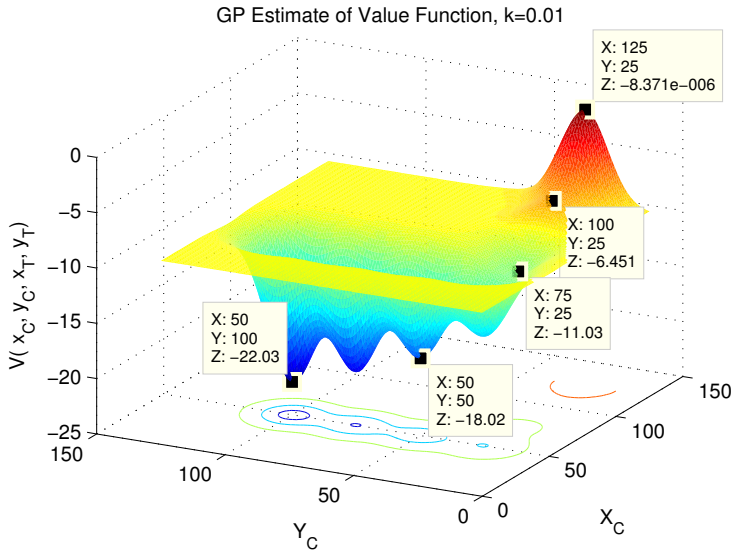


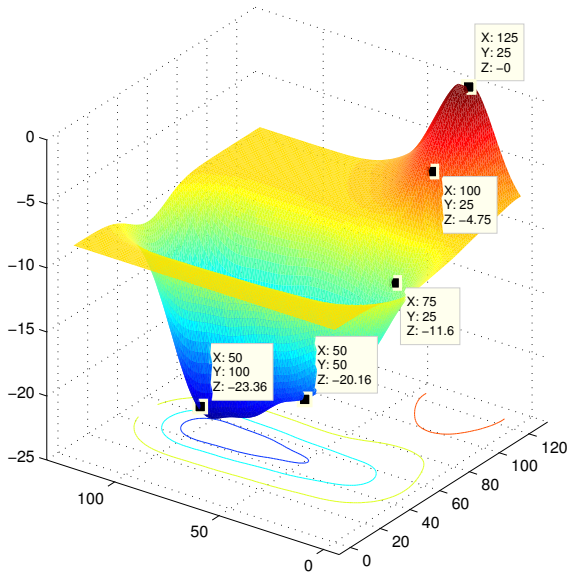
Figure 14: ***Parameterized*** V_{TM} ***vs. RAR and MAXQ:*** *These experiments compare the original Recursive Average Reward (RAR) and MAXQ (discounted reward) algorithms against the parameterized* $V_{TM}(TaxiLoc, \text{WP})$ *path learner.*

surrogate function, and there are a number of situations where we might want to use other models.

- An engineer or scientist might have a model that is known to work well with the data, and re-engineering it with kernels might be an unacceptable expense.
- Gaussian processes are inherently *homoskedastic* – variance is assumed to be the same in all parts of the space. Furthermore, the covariance matrix is stationary. Neither is a realistic assumption in many cases.
- Gaussian processes are slow to train on large data sets.
- Conversely, very small data sets often do not have enough information to reliably learn kernel parameters.
- The most common GP kernels – Gaussian and Matérn – assume the generating function is continuous, and work best when the space is low-dimensional.



(a) V_{TM} GP $k = 0.01$



(b) V_{TM} GP $k = 0.02$

Figure 15: ***GP Response Surface.*** *A small kernel value narrows the 'footprint' of an observation, whereas a larger k interpolates to the surrounding state space.*

Of course, there now exists a very large body of literature on GPs in the machine learning community, and many of these are well-known problems for which extensions to the GP model exist. For example, the homoskedasticity problem is discussed in [Le *et al.*, 2005], and Treed Gaussian Processes deal with stationarity and (to an extent) data set size [Gramacy, 2005]. However, this body of literature can be quite daunting to the non-specialist, and it may not be practical to implement and test multiple algorithms. Many applications start with an engineer who has a well-understood problem and an existing model of the data – we would like this person to be able to apply Bayesian optimization using the tools they already have at hand.

There isn't yet a truly "black box" framework for using Bayesian optimization with an arbitrary model. However, at a low level the algorithm modularizes nicely, and with a little bit of domain knowledge it is possible to replace components fairly easily. In this section, we will detail our efforts to do so with the popular and successful *Random Forest* model [Breiman, 2001], and in doing so, hopefully we can provide a template for others to follow.

7.1 The Random Forest Algorithm

The simplicity, high performance and relative easy of tuning have made Random Forests a very popular "black box" model for classification and regression, and they are frequently a part of statistical and data analysis toolkits such as Orange [Demsar and Zupan, 2004] and TMVA [Hocker *et al.*, 2007].

The Random Forest (RF) algorithm was introduced by Breiman [2001], building on earlier work of Ho [1998], as an evolution of bagging that uses an ensemble of decorrelated trees, with each tree trained on a sample of the data, and each node of each tree trained on a sample of the available features. A description, based on [Hastie *et al.*, 2009] is shown in Algorithm 3.

Algorithm 3 Random Forest Training

for $b = 1$ to B **do**
  Draw a bootstrap sample $\mathcal{D}_b$ of size $|\mathcal{D}|$ from $\mathcal{D}$ (with replacement).
  Grow a tree $\mathcal{T}_b$ from $\mathcal{D}_b$, by recursively repeating the following steps:
  while Termination condition is not met **do**
    Select m variables at random from the available variables.
    Pick the best variable/split point from the pool available.
    Split the node into two child nodes.
  end while
end for
Output the ensemble of trees $\{\mathcal{T}_b\}_1^B$.

Furthermore, we elected to look at Random Forests because they have a number of properties that make them distinct from GPs, or at least the "vanilla" GP model we describe in Section 2. For example, Random Forests:

- The generalization error is bounded, making Random Forests robust to noise and outliers.

- Work well with large data sets.
- Do not assume homoskedasticity or stationarity.
- Generally work quite well in both high- and low-dimensional spaces.
- Do not assume data features are continuous, or even real. Features need not even be numeric – Random Forests work well with categorical features.
- Out-of-bag error can be used to measure forest performance and set termination conditions without using cross-validation or held-out data (though at some cost to accuracy).

7.2 Implementation

There are a number of implementation decisions that need to be made.

- **Choice of m.** To preserve variance between trees, the RF algorithm requires each node be constructed by sampling possible features and picking the best one. This requires us to select m, the number of features to test. The choice of m affects performance, training time, and variance. It also indirectly affects tree size, which affects storage and test time. [Bosch *et al.*, 2007] suggest a schedule in which m is dependent on node depth – at the root, m is very small, to maximize variance. As depth increases, m increases, to improve performance and reduce tree complexity.
- **Split selection criteria.** A variety of criteria have been suggested for selecting which feature and split to select for each node, including Gini and information gain, as well as biasing schema such as Marshall correction.
- **Termination condition.** While it is possible to grow each tree until each leaf contains a single datum, this makes for very complex, slow forests. It is generally preferable to terminate based on either a maximum depth, or, more commonly, when the number of data in a node falls below some value $ndata$, typically 5.

Furthermore, a major advantage of using bootstrapping techniques is that each tree can be trained on a different sample of the data. While traditional bootstrapping samples N times from the data, uniformly, with repetition to create each training set $\mathbf{Z}_b$, this is only necessary to prove certain theoretical bounds. Empirically, we have observed that sampling M times, where $M < N$, can perform as well or better, and the sampling need not be uniform.

In active learning, this is very appealing, because it means we can sample from a much larger negative data set, and we can explicitly enforce balance in the training classes. We can also sample nonuniformly so that more important data (for example, more recent samples) are more likely to be selected.

7.3 Adapting Random Forests

One approach might be to develop a new acquisition function for the Random Forest model, but the $EI(\cdot)$ is fairly well-understood and had been shown to work well, so we wish to continue to use it. This means we need to define the mean, $\mu(\cdot)$, and variance, $\sigma^2(\cdot)$, of the posterior.

The mean is straightforward – we can use the random forest estimator,

$$\mu_{\texttt{rf}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\mathcal{T}_b(\mathbf{x}).$$

The variance is actually a little trickier. Initially, we used the variance of the predictions over the trees in the ensemble, which we called the "ensemble" variance:

$$\sigma^2_{ensemble}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}(\mu_{\texttt{rf}}(\mathbf{x}) - \mathcal{T}_b(\mathbf{x}))^2$$

However, this is intuitively unappealing, as it only measures tree disagreement. We would like to actually measure the variance of the fit of the data of each tree. To take this into account we instead use the ensemble variance of the leaves:

$$\sigma^2_{\texttt{rf}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{|\mathcal{D}_b^{\mathbf{x}}|}\sum_{f(\mathbf{x}_i)\in\mathcal{D}_b^{\mathbf{x}}}\left(\mu_{\texttt{rf}}(\mathbf{x}) - f(\mathbf{x}_i)\right)^2$$

where $\mathcal{D}_b^{\mathbf{x}}$ denotes the subset of $\mathcal{D}_b$ assigned to the same leaf as $\mathbf{x}$ in tree $\mathcal{T}_b$. Note that in the case where $ndata = 1$, $\sigma^2_{\texttt{rf}}$ collapses to $\sigma^2_{ensemble}$.

Armed with these, we can evaluate the acquisition function $EI_{\texttt{rf}}(\cdot)$ by substituting $\mu_{\texttt{rf}}(\cdot)$ and $\sigma^2_{\texttt{rf}}(\cdot)$ into Equation (5). However, in the case of $EI_{\texttt{rf}}$, both $\mu_{\texttt{rf}}(\cdot)$ and $\sigma^2_{\texttt{rf}}(\cdot)$ are sums of piecewise constant functions, so $EI_{\texttt{rf}}(\cdot)$ is also a piecewise constant function. We can thus optimize $EI_{\texttt{rf}}(\cdot)$ by evaluating it at each interval and taking the maximum. However, if the objective can be assumed to be reasonably well-behaved, it may still be preferable to use DIRECT, as it is substantially faster when a forest has many trees (and therefore many intervals).

7.4 model updates

An appealing feature of Random Forests is that they can be updated in an online fashion efficiently without retraining. One method that works well in practice is shown in Algorithm 4.

It is necessary to do some tree retraining in order to account for the fact that internal nodes might no longer be good splitting points as new data are added. In practice, it seems to work quite well to randomly retrain each tree with a probability between 0.01 and 0.1 each iteration. An alternative would be to retrain trees that fall under some error-reduction or correlation threshold.

Algorithm 4 Adding new data to a Random Forest

Input new $(\mathbf{x}^\star, f(\mathbf{x}^\star))$ pair.
for $b = 1$ to B **do**
  if $\mathcal{T}_b$ is retraining candidate **then**
    Retrain the tree from scratch, using a new bootstrap sample, $\mathcal{D}_b$.
  else
    Add $(\mathbf{x}^\star, f(\mathbf{x}^\star))$ to $\mathcal{D}_b$.
    Assign $(\mathbf{x}^\star, f(\mathbf{x}^\star))$ to the appropriate leaf of $\mathcal{T}_b$.
    if leaf no longer meets leaf conditions ($|\mathcal{D}_b^{\mathbf{x}^\star}| > ndata$ and $f(\mathbf{x}^\star)$ differs from leaf prediction, or $|\mathcal{D}_b^{\mathbf{x}^\star}| <= ndata$ and $f(\mathbf{x}^\star)$ would cause leaf label to change) **then**
      Split leaf and train children.
    end if
  end if
end for

To do active learning, we can use tree disagreement as a measure of uncertainty. A catch is that because the leaves of a Random Forest are defined over overlapping constant-value intervals, Random Forests cannot easily distinguish between regions that are highly uncertain, and regions that actually are well-understood but steep. When we do active learning, we want to sample from the former and avoid the latter. A solution in principle (*ie* one that I haven't actually tried) is to track the rate of change of disagreement. Regions where more samples don't result in improvement are likely areas that are difficult for Random Forests to model, and should be avoided until better areas are sampled first. If we assume that sample density is correlated with certainty, we could enforce this by adding a penalization term to the uncertainty based on sample density, to drive sampling toward areas with few labels.

7.5 Comments and Observations

Figure 16 shows a toy examples to illustrate some of the differences between GP-based and RF-based Bayesian optimization:

- In the GP model, $\sigma^2(\cdot)$ is entirely due to the distance a point is from the observed values. The shape of the function plays no role. In the RF model, $\sigma^2_{\texttt{rf}}(\cdot)$ is also based on tree disagreement. As a result, $\sigma^2_{\texttt{rf}}(\cdot)$ tends to be high where $f(\cdot)$ is steep.

- Similarly, while sampling in the GP model is guaranteed to reduce variance, variance is only decreased in the RF model when the leaves of different trees have low variance – this usually happens in regions that are fairly flat.

- In the GP model, the surrogate function tends to run through the samples values. In the RF model, this is not the case, because each sample is present in only $\frac{1}{e}$ of the tree training sets, and $\mu_{\texttt{rf}}(\cdot)$ is the mean of all the trees. As a result, the RF model is known to be highly resistant to noise, outliers and overfitting, but at the cost of approximation accuracy.

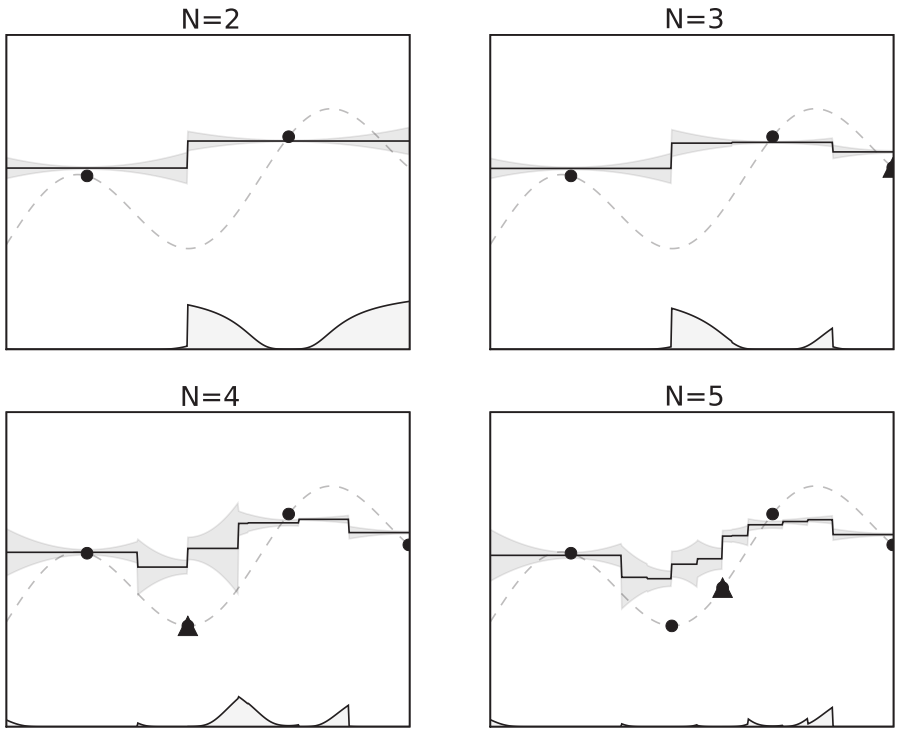


Figure 16: *Random forest prior on a toy 1D function, analogous to the GP case of Figure 1. Note that in the sampling in the steep regions of the function decreases variance only slowly, as the region is difficult to fit with sums of piecewise constant functions. Also note the relatively slow change in the surrogate function as more points are added – random forests are much more resistant to dramatic changes based on small amounts of evidence.*

- Generally speaking, it takes more data to change a random forest model than a Gaussian process model. A single datum can radically reconfigure a GP, but the bootstrapping and random subspaces of an RF means that a great deal more evidence must be accumulated.

In general, we would not suggest using an RF as the surrogate function where a GP would be expected to work well. However, in those cases where the data or application do not lend themselves well to the GP framework, it may well be worth trying an RF surrogate. Based on our research, this might tend to happen when the data is high-dimensional, the function is not smooth, or it is not desirable for the surrogate to change dramatically between iterations.

8 Discussion

Bayesian optimization is a powerful tool for machine learning, where the problem is often not acquiring data, but acquiring labels. In many ways, it is like conventional active learning, but instead of acquiring training data for classification or regression, it allows us to develop frameworks to efficiently solve novel kinds of learning problems such as those discussed in Sections 5 and 6. It proves us with an efficient way to learn the solutions to problems, and to collect data, all within a Bayesian framework.

However, Bayesian optimization is also a fairly recent addition to the machine learning community, and not yet extensively studied. Here, we wish to describe some of the shortcomings we have experienced in our work with Bayesian optimization, both as caveats and as opportunities for other researchers.

A particular issue is that the design of the prior is absolutely critical to efficient Bayesian optimization. As we discussed in Section 7, Gaussian processes are not always the best or easiest solution, but even when they are, great care must be taken in the design of the kernel. In many cases, though, little is known about the objective function, and, of course, it is expensive to sample from (or we wouldn't need to use Bayesian optimization in the first place). The practical result is that in the absence of (expensive) data, either strong assumptions are made without certainty that they hold, or a weak prior must be used. It is also often unclear how to handle the trade-off between exploration and exploitation in the utility function. Too much exploration, and many iterations can go by without improvement. Too much exploitation leads to local maximization.

These problems are exacerbated as dimensionality is increased – more dimensions means more samples are required to cover the space, and more parameters and hyperparameters may need to be tuned, as well. In order to deal with this problem effectively, it may be necessary to do automatic feature selection, or assume independence and optimize each dimension individually.

This makes Bayesian optimization anything but a "black box". A good understanding of both the objective and the prior is necessary to employ it successfully. While this is not at all new to machine learning or optimization, Bayesian optimization currently lacks the science and engineering tools that areas like classification have, where a variety of surveys and data analysis programs exist to help non-specialists apply the method to a real-world problem.

There are many extensions that will need to be made to Bayesian optimization for particular applications – feature selection, time-varying models, censored data, heteroskedasticity, non-stationarity, non-Gaussian noise, *etc.* In many cases, these can be dealt with as extensions to the prior – in the case of Gaussian processes, for example, a rich body of literature exists in which such extensions have been proposed. However, these extensions need to take into account the adaptive and iterative nature of the optimization problem, which can vary from trivial to impossible.

Although we focused on expensive objective functions in this tutorial, there is no reason to believe that Bayesian optimization could not be applied to other problems. If the cost function is relatively easy to evaluate, but has many local

minima that must be avoided, then a global optimization scheme might provide a more reasonable solution than gradient methods. Moreover, in cases where the cost function is not differentiable, for example some forms of cross-validation, Bayesian optimization also holds promise and should be investigated. Finally, there is no literature on Bayesian optimization methods for large dimensional search spaces. However, if the objective function is easy to evaluate and multimodal, we conjecture that these methods might provide reasonable solutions even in high dimensions.

Clearly, there is a lot of work to be done in Bayesian optimization, but we feel that the doors it opens make it worthwhile. It is our hope that as Bayesian optimization proves itself useful in the machine learning domain, the community will embrace the fascinating new problems and applications it opens up.

References

[Andre, 2003] D. Andre. *Programmable Reinforcement Learning Agents*. PhD thesis, University of California at Berkley, 2003.

[Audet *et al.*, 2000] C. Audet, J. Jr, Dennis, D. W. Moore, A. Booker, and P. D. Frank. Surrogate-model-based method for constrained optimization. In *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[Barto and Mahadevan, 2003] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[Bartz-Beielstein *et al.*, 2005] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proc. CEC-05*, 2005.

[Baxter *et al.*, 2001] J. Baxter, P. L. Bartlett, and L. Weave. Experiments with infinite-horizon, policy-gradient estimation. *J. Artificial Intelligence Research*, 15:351–381, 2001.

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Bosch *et al.*, 2007] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *ICCV*, 2007.

[Boyle, 2007] P. Boyle. *Gaussian Processes for Regression and Optimisation*. PhD thesis, Victoria University of Wellington, Wellington, New Zealand, 2007.

[Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[Brochu and de Freitas, 2006] E. Brochu and N. de Freitas. Active learning as interactive design. Technical Report UBC TR-2006-7, University of British Columbia, 2006.

[Brochu *et al.*, 2007] E. Brochu, N. de Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems*, 2007.

[Chu and Ghahramani, 2005a] W. Chu and Z. Ghahramani. Extensions of Gaussian processes for ranking: semi-supervised and active learning. In *Learning to Rank workshop at NIPS-18*, 2005.

[Chu and Ghahramani, 2005b] W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *ICML*, 2005.

[Cora, 2008] V. M. Cora. Model-based active learning in hierarchical policies. Master's thesis, University of British Columbia, Vancouver, Canada, April 2008.

[De Grave *et al.*, 2008] K. De Grave, J. Ramon, and L. Raedt. Active learning for high throughput screening. In *International Conference on Discovery Science*, pages 185–196. Springer-Verlag, 2008.

[Demsar and Zupan, 2004] J. Demsar and B. Zupan. Orange: From experimental machine learning to interactive data mining. White paper, University of Ljubljana, 2004.

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

[Elder, 1992] J. F. Elder, IV. Global R^d optimization when probes are expensive: The GROPE algorithm. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 1992.

[Élő, 1978] Á. Élő. *The Rating of Chess Players: Past and Present*. Arco Publishing, New York, 1978.

[Genton, 2001] M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[Ghavamzadeh, 2005] M. Ghavamzadeh. *Hierarchical Reinforcement Learning in Continuous State and Multi-agent Environments*. PhD thesis, University of Massachusetts Amherst, 2005.

[Ginsbourger *et al.*, 2008] D. Ginsbourger, R. Le Riche, and L. Carraro. A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes. 2008.

[Glickman and Jensen, 2005] M. E. Glickman and S. T. Jensen. Adaptive paired comparison design. *Journal of Statistical Planning and Inference*, 127:279–293, 2005.

[Gramacy, 2005] R. B. Gramacy. *Bayesian Treed Gaussian Process Models*. PhD thesis, University of California, Santa Cruz, December 2005.

[Hastie *et al.*, 2009] T. Hastie, R. Tibrishani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, second edition, 2009.

[Herbrich and Graepel, 2006] R. Herbrich and T. Graepel. Trueskill: A Bayesian skill rating system. Technical Report MSR-TR-2006-80, Microsoft Research, June 2006.

[Hertzmann, 2003] A. Hertzmann. Machine learning for computer graphics: A manifesto and tutorial. In *Pacific Graphics*, 2003.

[Hinton and Salakhutdinov, 2006] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[Ho, 1998] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), 1998.

[Hocker *et al.*, 2007] A. Hocker, P. Speckmayer, J. Stelzer, F. Tegenfeldt, H. Voss, and K. Voss. Tmva - toolkit for multivariate data analysis. Technical Report arXiv:physics/0703039v4, arXiv, 2007.

[Huang *et al.*, 2006] D. Huang, T. T. Allen, W. I. Notz, and N. Zheng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466, March 2006.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proc. GECCO'09*, 2009.

[Jones *et al.*, 1993] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[Jones *et al.*, 1998] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[Jones, 2001] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[Krige, 1951] D. G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, 1951.

[Kushner and Yin, 1997] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications*. Springer-Verlag, 1997.

[Kushner, 1962] H. J. Kushner. Stochastic model of an unknown function. *Journal of Mathematical Analysis and Application*, 5:150–167, 1962.

[Kushner, 1964] H. J. Kushner. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[Le *et al.*, 2005] Q. V. Le, A. J. Smola, and S. Canu. Heteroscedastic Gaussian process regression. In *IJCAI*, 2005.

[Ledda *et al.*, 2005] P. Ledda, A. Chalmers, T. Troscianko, and H. Seetzen. Evaluation of tone mapping operators using a high dynamic range display. In *SIGGRAPH*, August 2005.

[Lizotte *et al.*, 2007] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans. Automatic gait optimization with gaussian process regression. In *IJCAI*, 2007.

[Lizotte, 2008] D. Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 2008.

[Locatelli, 1997] M. Locatelli. Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization*, 1997.

[Marks *et al.*, 1997] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. *Computer Graphics*, 31, 1997.

[Marthi *et al.*, 2005] B. Marthi, D. Latham, S. Russell, and C. Guestrin. Concurrent hierarchical reinforcement learning. *In Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[Martinez-Cantin *et al.*, 2006] R. Martinez-Cantin, N. de Freitas, and J. A. Castellanos. Analysis of particle methods for simultaneous robot localization and mapping and a new algorithm: Marginal-SLAM. In *Proc. IEEE International Conference on Robots and Automation*, 2006.

[Martinez-Cantin *et al.*, 2007] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems (RSS)*, 2007.

[Martinez-Cantin *et al.*, 2009] R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 2009.

[McFadden, 2001] D. McFadden. Economic choices. *The American Economic Review*, 91:351–378, 2001.

[Mockus *et al.*, 1978] J. Mockus, V. Tiesis, and A. Žilinskas. *Toward Global Optimization*, volume 2, chapter The Application of Bayesian Methods for Seeking the Extremum, pages 117–128. Elsevier, 1978.

[Mockus, 1994] J. Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347 – 365, 1994.

[Mosteller, 1951] F. Mosteller. Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. *Psychometrika*, 16:3–9, 1951.

[Ng and Jordan, 2000] A. Y. Ng and M. I. Jordan. Pegasus: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence (UAI2000)*, 2000.

[Ngan *et al.*, 2006] A. Ngan, F. Durand, and W. Matusik. Image-driven navigation of analytical BRDF models. In T. Akenine-Möller and W. Heidrich, editors, *Eurographics Symposium on Rendering*, 2006.

[Parr, 1998] R. E. Parr. *Hierarchical control and learning for markov decision processes*. PhD thesis, 1998. Chair-Stuart Russell.

[Poyiadjis *et al.*, 2005] G. Poyiadjis, A. Doucet, and S. S. Singh. Particle methods for optimal filter derivative: Application to parameter estimation. In *IEEE ICASSP*, pages 925–928, 2005.

[Santner *et al.*, 2003] T. J. Santner, B. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[Sasena, 2002] M. J. Sasena. *Flexibility and Efficiency Enhancement for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, 2002.

[Schonlau *et al.*, 1998] M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, 34:11–25, 1998.

[Schonlau, 1997] M. Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[Shoham and Leyton-Brown, 2009] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[Snelson *et al.*, 2003] E. Snelson, C. E. Rasmussen, and Z. Ghahramani. Warped Gaussian processes. In *NIPS*, 2003.

[Stam, 2003] J. Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*, 2003.

[Stern, 1990] H. Stern. A continuum of paired comparison models. *Biometrika*, 77:265–273, 1990.

[Stuckman, 1988] B. Stuckman. A global search method for optimizing nonlinear systems. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):965–977, 1988.

[Sutton *et al.*, 1999] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Sutton, 1998] R. S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Thurstone, 1927] L. Thurstone. A law of comparative judgement. *Psychological Review*, 34:273–286, 1927.

[Vasquez and Bect, 2008] E. Vasquez and J. Bect. On the convergence of the expected improvement algorithm. *ArXiv e-prints*, (0712.3744v2), Feb 2008.

[von Neumann and Morgenstern, 1947] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, 1947.

[Wymann *et al.*, 2009] B. Wymann, C. Dimitrakakis, and C. A. et al. The open racing car simulator (http://torcs.sourceforge.net/), 2009.

[Younes, 1989] L. Younes. Parameter estimation for imperfectly observed Gibbsian fields. *Prob. Theory and Rel. fields*, 82:625–645, 1989.

[Žilinskas and Žilinskas, 2002] A. Žilinskas and J. Žilinskas. Global optimization based on a statistical model and simplical partitioning. *Computers and Mathematics with Applications*, 44:957–967, 2002.