BO Extensions & Variants

Up until now, we have only considered a relatively simple version of Bayesian optimization (BO): a sequential optimization procedure where observations of a single, noiseless objective function are made one by one until some budget has been expended. These idealized assumptions make for a convenient presentation/baseline for study but many real-world optimization problems do not fit into this neat little mold. As such, researchers have formulated many extensions and variants of this prototypical BO routine. We will briefly present some of the most relevant/well-studied of these.

Despite their seeming complexity, our approach to solving these extensions and variants will follow the same rough framework that we have previously developed:

1. Define the action space $\mathcal{A}$ for each setting.
2. Specify preferences over outcomes in terms of a utility function, $u(\mathcal{D})$.
3. Identify the uncertain or variable components and determine how they relate to observations of the objective function $p(\psi \mid \mathcal{D})$.
4. Compute the action that maximizes the expected marginal utility, $a^* \in \mathcal{A}$.
5. Define an optimal policy based on this one-step optimal action.

Cost-aware Optimization

One implicit assumption of the BO routine described above is that each observation "costs" the same amount or counts as the same expenditure against our finite budget: given a budget of T observations, we can observe the objective function at precisely T locations. In practice, it is often the case that different locations in the domain have different associated costs to observe. For example, consider the task of hyperparameter optimization: computing the validation error for certain settings of the model's hyperparameters (e.g., small step sizes, large number of layers/dimensionality) might take much longer/require more compute resources than others. As such, it could make sense to incorporate some notion of these varying costs when deciding where to observe the objective function next.

We will consider two different settings for this variant: known costs, where the cost of evaluating the objective everywhere in the domain is known a priori (or can be well approximated), and unknown costs, where the cost of an observation is revealed only after selecting the location.

Known Costs

In the case where the cost of each evaluation is known in advance, it is relatively straightforward to incorporate this information into the acquisition function via a "cost-benefit" style analysis. Formally, suppose that the cost of observing the objective function at location $\mathbf{x}$ is given by

$$c(\mathbf{x}) \; \forall \; \mathbf{x} \in \mathcal{X}.$$

Further, assume that these costs are additive (a reasonable assumption given the sequential nature of BO) such that the cost of a set of observations, $\mathcal{D} = \{(\mathbf{x}_i, f_i = f(\mathbf{x}_i))\}_{i=1}^n$ is simply

$$c(\mathcal{D}) = \sum_{i=1}^{n} c(\mathbf{x}_i).$$

If the utility of a dataset and its costs can be put in roughly the same units (e.g., dollars via conversion into monetary gains/costs), then we can simply define a *cost-aware* dataset utility as

$$v(\mathcal{D}) = u'(\mathcal{D}) - c(\mathcal{D}) \text{ where } u'(\mathcal{D}) \text{ is the dataset utility function.}$$

From here, we can continue as before and compute the point-wise utility as the marginal change in utility for observing the function at some location:

$$\begin{aligned} u(\mathbf{x}) &= v\Big(\mathcal{D} \cup \big(\mathbf{x}, f(\mathbf{x})\big)\Big) - v(\mathcal{D}) \\ &= u'\Big(\mathcal{D} \cup \big(\mathbf{x}, f(\mathbf{x})\big)\Big) - c\Big(\mathcal{D} \cup \big(\mathbf{x}, f(\mathbf{x})\big)\Big) - u'(\mathcal{D}) + c(\mathcal{D}) \\ &= \left[u'\Big(\mathcal{D} \cup \big(\mathbf{x}, f(\mathbf{x})\big)\Big) - u'(\mathcal{D})\right] - c(\mathbf{x}). \end{aligned}$$

The first term corresponds to the marginal improvement in the dataset utility, which we have already seen translated into various acquisition functions for different dataset utility functions. All we do to incorporate known costs is subtract the associated cost from this marginal gain giving rise to the cost-aware acquisition function

$$a_{\text{CA}}(\mathbf{x}) = \mathbb{E}\big[u(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}\big] = \mathbb{E}\left[u'\Big(\mathcal{D} \cup \big(\mathbf{x}, f(\mathbf{x})\big)\Big) - u'(\mathcal{D})\right] - c(\mathbf{x}) \text{ (assuming deterministic costs).}$$

Figure 1 shows the expected improvement (EI) acquisition function on our running example, adjusted by a linearly increasing cost over the domain. Note that the dataset utility is always strictly non-negative but the cost-adjusted acquisition function can take on negative values.

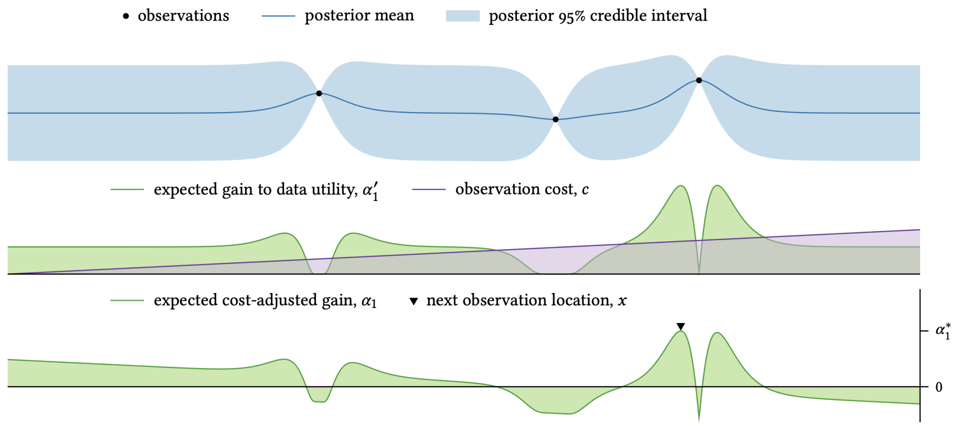


Figure 1: Cost-aware expected improvement where the cost of making an observation increases as x increases, as indicated by the upward sloping purple line (middle); the cost-adjusted EI (bottom) is simply the difference of the green and purple curves.

Unknown Costs

When the observation costs are unknown a priori, a natural approach is to treat the cost function, $c : \mathcal{X} \to \mathbb{R}$, as another black-box function to be learned or approximated in an online manner. Intuitively, each time we select a location to observe, we observe the cost at that location in addition to the objective function's value, $\mathcal{D} = \{(\mathbf{x}_i, f_i = f(\mathbf{x}_i), c_i = c(\mathbf{x}_i))\}_{i=1}^{n}$.

If we assume that the costs are conditionally independent of the objective function given $\mathbf{x}$, we can model c using a separate, independent probabilistic process (a Gaussian process perhaps?), allowing us to account for non-deterministic costs. We then simply apply the same cost-aware framework above except we treat the cost term as a random variable to be included inside the expectation.

Figure 2 again shows the EI acquisition function on our running example, this time with a learned cost function, which was assumed to follow a Gaussian process belief; note that even though the posterior distribution of the cost is represented, because the cost and objective function are assumed to be independent, the cost-adjusted acquisition function is simply shifted by the posterior mean.

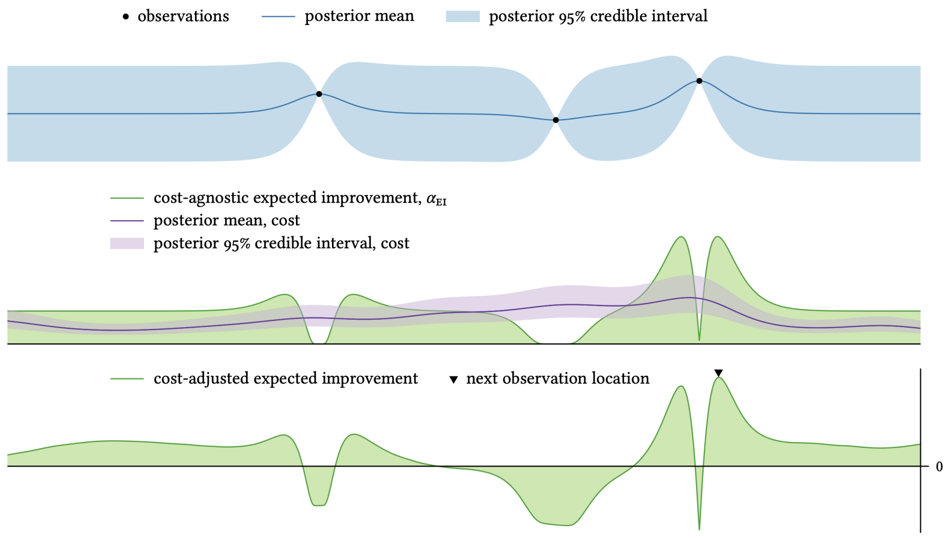


Figure 2: Cost-aware expected improvement where the cost of making an observation is learned via Gaussian process inference; the posterior Gaussian process belief on the cost function is shown in purple (middle). The cost-adjusted EI (bottom) is still the difference between the green and purple curves as the expected cost is equal to the posterior mean.

If the cost is not independent of the objective function, then the two can be modeled using a *joint* probabilistic process, which defines a joint probability distribution over f_i and c_i, $p(f_i, c_i \mid \mathbf{x}_i, \mathcal{D})$. This joint distribution can be used to evaluate the necessary expected utilities and thus, derive the corresponding cost-aware acquisition function.

Joint Gaussian Processes

Intuitively, recall that a GP belief on a single function can be thought of as an infinite-dimensional multivariate Gaussian. Under this interpretation, it is trivial to simply "stack" another function on top of this existing GP; after all, $\infty + \infty$ is still ∞!

Formally, given two functions, $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$, defined over the same domain, joint GP belief over f and g can be written as

$$p(f, g) = \mathcal{GP}\left(\begin{bmatrix} f \\ g \end{bmatrix}; \begin{bmatrix} \mu_f \\ \mu_g \end{bmatrix}, \begin{bmatrix} k_f & k_{fg} \\ k_{gf} & k_g \end{bmatrix}\right).$$

Note that under this definition, both f and g follow marginal GP beliefs:

$$p(f) = \mathcal{GP}(f; \mu_f, k_f) \text{ and } p(g) = \mathcal{GP}(g; \mu_g, k_g).$$

The key consideration in defining a joint GP is the design of the cross-covariance functions k_{fg} and k_{gf}. These define how observations of one function are related to observations of the other:

$$\begin{aligned} k_{fg}(\mathbf{X}, \mathbf{X}') &= \operatorname{cov}\left(f(\mathbf{X}), g(\mathbf{X}')\right) \\ k_{gf}(\mathbf{X}', \mathbf{X}) &= \operatorname{cov}\left(g(\mathbf{X}'), f(\mathbf{X})\right) = k_{fg}(\mathbf{X}, \mathbf{X}')^T. \end{aligned}$$

Figure 3 shows the (marginal) posterior beliefs of f and g using a joint GP belief conditioned on 5 observations of each function. This model uses the standard squared exponential kernel for both k_f and k_g and sets $k_{fg} = k_{gf} = 0.9k_f$; this implies a strong correlation between the two functions at every location: $\operatorname{corr}(f(x), g(x)) = 0.9$. This can be seen in the posterior beliefs: even at locations only indirectly observed on the other function, there is a significant reduction in the posterior variance and the posterior mean skews strongly to fit those indirect observations.

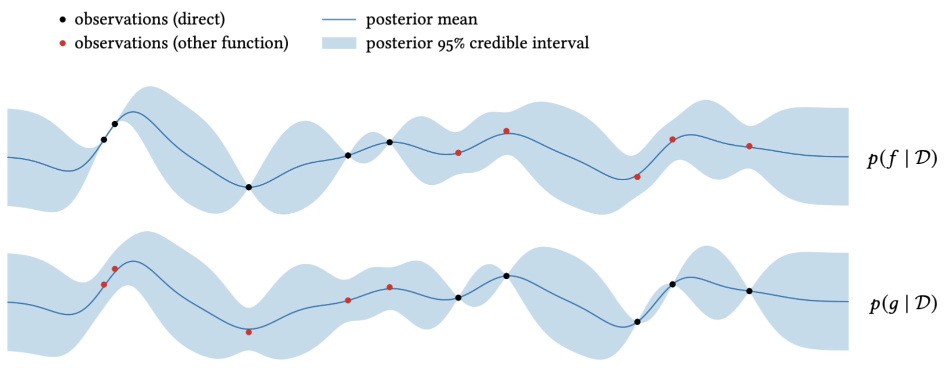


Figure 3: A joint GP belief over two functions, f and g, conditioned on 10 observations: 5 observations of f and 5 observations of g. Using the strong cross-covariance functions defined in main text, there is a clear impact on the posterior belief about f at locations where we observe g and vice versa.

Batch Optimization

Parallelization is a frequently used approach to scale up sequential processes, particularly when the limiting factor in gathering observations is time. Returning to our motivating example of hyperparameter optimization, practitioners typically test many hyperparameter settings simultaneously across multiple machines/processors so that one experiment is not bottle-necking all the others.

Formally, in each iteration of *batch* Bayesian optimization, we select a set of b locations, $\mathbf{X}_t = \{\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(2)}, \ldots, \mathbf{x}_t^{(b)}\}$, and observe their corresponding objective function values, $\mathbf{f}_t = \{f_t^{(1)}, f_t^{(2)}, \ldots, f_t^{(b)}\}$. In particular, for the purposes of these notes, we will only consider the *synchronous* batch setting, where all experiments in the current batch must complete before any of the next batch's locations can be chosen; the alternative is known as the *asynchronous* batch setting, where we can select a new location to observe as soon as any of the current batch's experiments finish. While interesting in its own right, the asynchronous setting is quite complicated and beyond the scope of this course.

Returning to our formalization of BO as a decision problem, the action space for the batch setting is $\mathcal{A} = \mathcal{X}^b$. If $u'(\mathcal{D})$ is the dataset utility, we can once again define the batch acquisition function as the expected improvement in the dataset utility for observing the set of locations $\mathbf{X}$:

$$\begin{aligned}\beta(\mathbf{X}) = \mathbb{E}\big[u(\mathbf{X}) \mid \mathbf{X}, \mathcal{D}\big] &= \mathbb{E}\Bigg[u'\Big(\mathcal{D} \cup \big(\mathbf{X}, \mathbf{f}\big)\Big) - u'(\mathcal{D})\Bigg] \\ &= \int u'\Big(\mathcal{D} \cup \big(\mathbf{X}, \mathbf{f}\big)\Big) p(\mathbf{f} \mid \mathbf{X}, \mathcal{D}) \, d\mathbf{f} \; - u'(\mathcal{D})\end{aligned}$$

Unfortunately (and unsurprisingly), the multidimensional integral in the acquisition function above is intractable for many of the dataset utility functions we have seen. Furthermore, even if we are able to approximate this expectation, actually optimizing the acquisition function can be challenging given the high-dimensional action space. As such, researchers have developed strategies to either

- extend specific acquisition functions to the batch setting or
- derive generalizable approximations to the batch acquisition function.

We briefly highlight some of these efforts, prioritizing the simpler techniques for the sake of brevity.

Batch Thompson Sampling

Perhaps the easiest acquisition function to extend to the batch setting is Thompson sampling: for a batch size of b, we can simply draw b sample paths from the posterior GP belief on f (instead of just 1 as we had done previously) and form a batch out of the b locations that maximize each sample path. Indeed, this simple approach even enjoys some nice theoretical results: Kandasamy et al. (2018) were able to bound the regret of this batch acquisition function in both the synchronous and asynchronous batch settings.

Batch Expected Improvement

Surprisingly, the batch acquisition function above is computable in closed form for the EI dataset utility function! However, computing it exactly requires b evaluations of a b-dimensional Gaussian CDF and b^2 evaluations of $(b-1)$-dimensional Gaussian CDF; while we have great approximations for low-dimensional Gaussian CDFs, once we move beyond say 4 dimensions, the quality of these approximations degrades significantly. Empirically, exactly computing this acquisition function is only feasible for relatively small batch sizes e.g., $b < 10$.

Figure 4 shows the (exact) EI acquisition function for a batch size of 2 on our running example. The optimal batch of size 2 consists of observations on either side of the current highest observed objective function value, a rather exploitative batch.

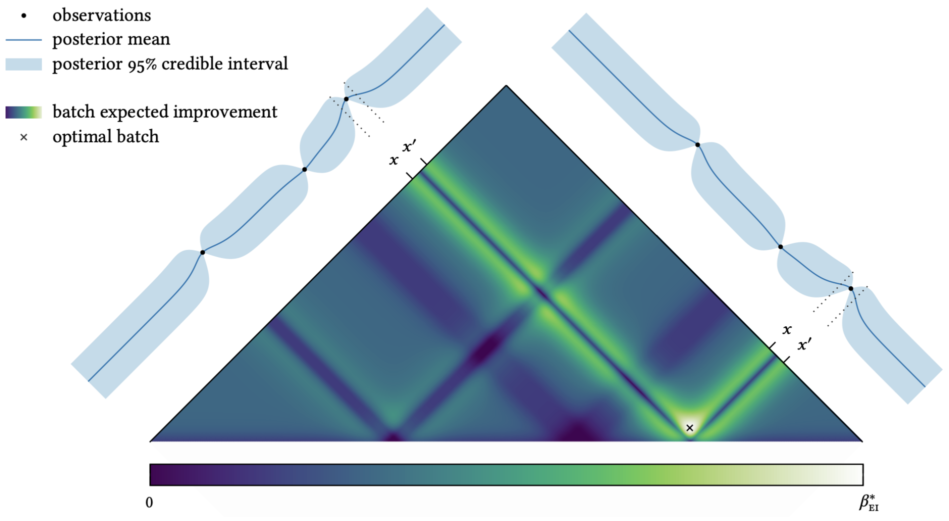


Figure 4: Batch expected improvement; the two axes each correspond to the location of one point in the batch and the posterior belief is shown duplicated along each axis. The grid indicates the expected improvement of observing a batch of two points for every possible pair of locations over the domain; the grid is symmetric so for the sake of presentation, only half is shown.

Sequential Simulation

A general strategy for approximating the optimal batch for any acquisition function is to decompose the construction of a batch into a sequential set of decisions. Given a sequential acquisition function a, we compute the first member of the batch by simply maximizing the acquisition function:

$$\mathbf{x}_t^{(1)} = \operatorname*{arg\,max}_{\mathbf{x} \in \mathcal{X}} a(\mathbf{x} \mid \mathcal{D}).$$

In order to figure out which location we would add to our batch next, we need to have some way of guessing the objective function's value at $\mathbf{x}_t^{(1)}$; call our imputed guess $\hat{f}_t^{(1)}$. There are a variety of strategies for setting $\hat{f}_t^{(1)}$:

- the *kriging believer* strategy uses $\hat{f}_t^{(1)} = \mu_{\mathcal{D}}(\mathbf{x}_t^{(1)})$ i.e., the posterior mean.
- the *constant liar* strategy simply sets $\hat{f}_t^{(1)} = c_t$ for some constant that is fixed for the entire batch; some common choices for c_t include the maximum, minimum and mean of the previously observed objective function values.

Regardless of how $\hat{f}_t^{(1)}$ is set, we now have a fictitious observation $(\mathbf{x}_t^{(1)}, \hat{f}_t^{(1)})$. Sequential simulation adds this data point to $\mathcal{D}$ and selects the second point in the batch using this augmented dataset:

$$\mathbf{x}_t^{(2)} = \operatorname*{arg\,max}_{\mathbf{x} \in \mathcal{X}} a\left(\mathbf{x} \mid \mathcal{D} \cup \left(\mathbf{x}_t^{(1)}, \hat{f}_t^{(1)}\right)\right).$$

We then guess another the objective function's value at $\mathbf{x}_t^{(2)}$ and continue until b points have been selected. In effect, sequential simulation approximately solves the bd-dimensional optimization problem by decomposing it into b smaller, d-dimensional optimizations.

Figure 5 shows the batch of 7 points which approximately optimizes the batch EI acquisition function, again on our running example. The 7 points demonstrate a nice trade-off between exploration and exploitation, with the first few points added to the batch being near existing, high objective function observations while the later points are more spread out over the domain.

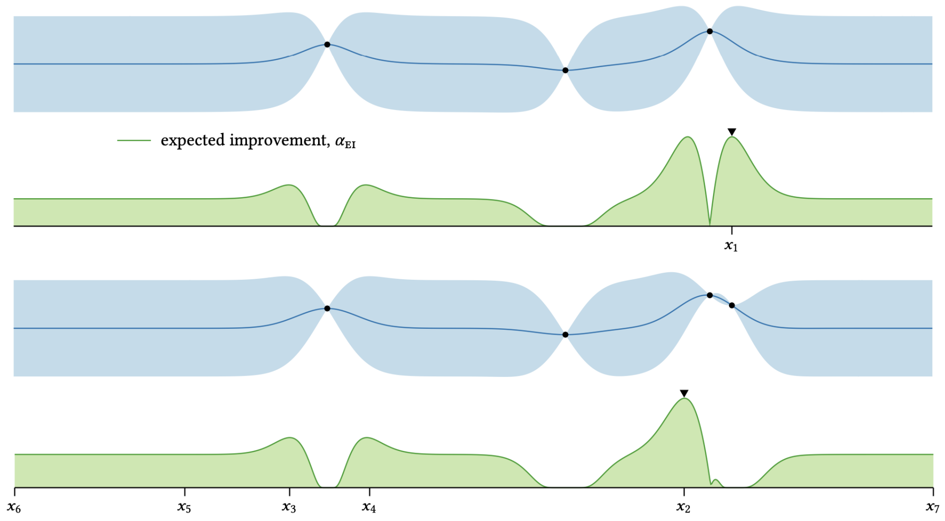


Figure 5: Approximate batch expected improvement, where the batch is computed using sequential simulation. The first point in the batch is exactly the point that sequential EI would observe next (second row). The kriging believer imputation strategy is used to impute the intermediate observations (thrid row).

Multifidelity Optimization

In many scientific discovery problems, practitioners often have access to inexpensive, low-quality *surrogates* of the objective function. For example, in robotics, scientists will typically first run computer simulations of an experiment to get a sense for how some policy will behave before testing it in the real-world. When optimizing machine learning hyperparameters, early termination can give you a poor but fast estimate of the validation error of some hyperparameter setting.

Formally, let f_* denote our true (but expensive to evaluate) objective function and suppose there are m cheaper, low-fidelity surrogate functions $\{f_1, f_2, \ldots, f_m\}$. Our action space in this variant consists of choosing both a function to observe and a location to observe the chosen function at: $\mathcal{A} = \{*, 1, 2, \ldots, m\} \times \mathcal{X}$.

To perform inference, we need to specify how an observation of a surrogate, $f_j(\mathbf{x})$ informs our belief about f_*. To do so, we can again leverage a joint GP over the functions $\{f_*, f_1, \ldots, f_m\}$. Specifying all the cross-covariance functions can be tedious although doing so is often necessary as observations of one surrogate might affect our belief about other surrogates as well as the objective function. A common choice is to assume the cross-covariance is a product kernel of the form:

$$k_{f_a f_b}(\mathbf{x}, \mathbf{x}') = k_{\mathcal{X}}(\mathbf{x}, \mathbf{x}') k_m(a, b)$$

for some fixed input covariance function $k_{\mathcal{X}}$ and some surrogate covariance k_m that is defined over the set $\{*, 1, 2, \ldots, m\}$. Finally, we assume that each function has some associated cost of observation, $\{c_*, c_1, \ldots, c_m\}$, which much like our discussion above, can be known or unknown or even correlated across surrogates. Regardless of how we choose to model these costs, the techniques from the cost-aware subsection above can be applied to this setting to determine which function is the most cost effective to observe in each iteration.

Figure 6 shows the cost-adjusted EI acquisition function for f^* and a single surrogate, f_1. We use the separable cross-covariance defined above, $k_{\mathcal{X}}$ is a squared exponential kernel, $k_m(*, *) = k_m(1, 1) = 1$ and $k_m(*, 1) = 0.8$. Furthermore, the cost of observing both functions is assumed to be constant, with f_1 being 10 times cheaper to observe than f_*. The cost-aware acquisition function in this case chooses to continue observing the surrogate.

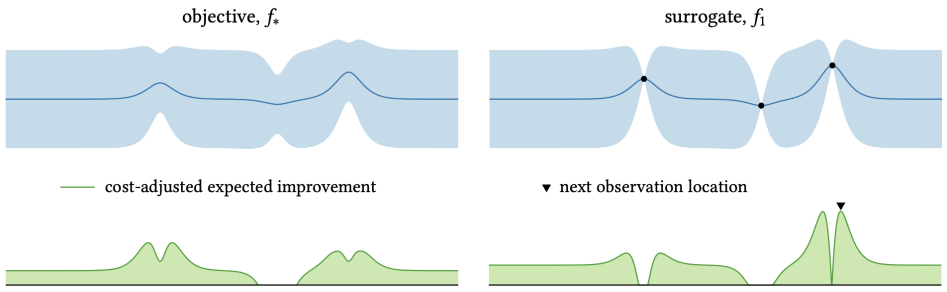


Figure 6: A joint GP belief over two functions, f and g, conditioned on 10 observations: 5 observations of f and 5 observations of g. Using the strong cross-covariance functions defined in main text, there is a clear impact on the posterior belief about f at locations where we observe g and vice versa.

Figure 7 shows a run of multifidelity BO in this setting. In total, 32 observations of f_1 were made and just 10 observations of f_* were made before the optimal value of f_* was located. Remarkably, almost all of the observations of f_* were made at locations where f_* takes on a (relatively) high value. This is because the joint GP belief was able to leverage the high correlation between f_* and f_1 and rule out regions where f_* was likely to be low, based on observations of f_1.

Multitask & Multiobjective Optimization

Finally, we will consider a pair of related extensions that are both concerned with settings where there are multiple objective functions, $\{f_1, f_2, \ldots, f_m\}$. Multitask optimization assumes that these functions are a set of related optimization tasks to be optimized individually, whereas multiobjective optimization requires that we identify a single location, $\mathbf{x}$, that simultaneously "optimizes" all m functions. Like before, a common (but not required) model in these settings is a joint GPs over the objective functions. Below, we will formally define these two extensions of BO and briefly describe some of the approaches used to solve them.

Multitask BO

As an example of multitask optimization, suppose we have an online recommender system that serves multiple customers who share certain preferences: each individual can be associated with a separate but related objective function and the goal is to provide each customer the best, individualized recommendation. This task is related to transfer learning: by transferring information across objective functions, we can (hopefully) substantially speed up their collective optimization.

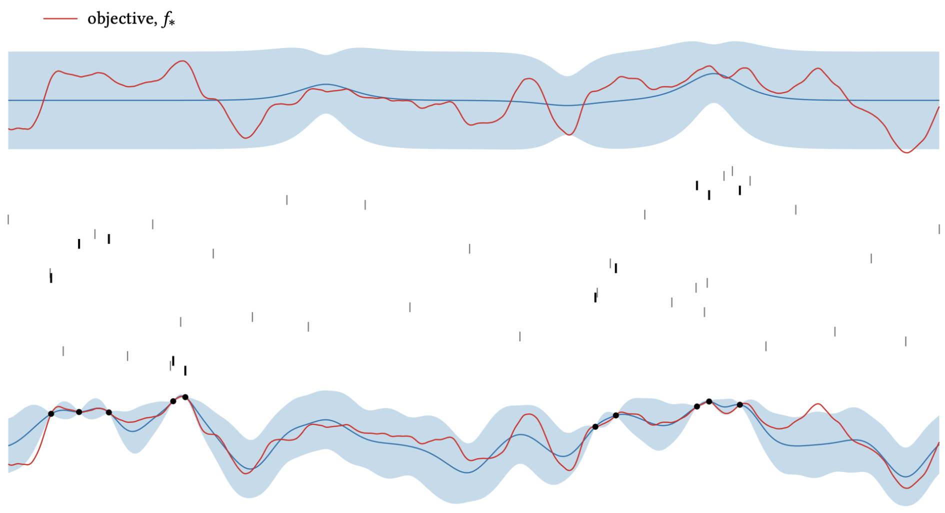


Figure 7: Multifidelity BO in the setting described in the main text. Thinner hash marks indicate the location of observations made of the surrogate while thicker hash marks indicate observations made of f_*; the relative height of the hash marks indicate the order in which observations were made.

As noted above, we will model the set of functions $\{f_1, f_2, \ldots, f_m\}$ using a joint GP. There are two ways that the multitask setting can be formulated: the objective functions can be considered *sequentially* i.e., in each iteration, we only consider a single function f_j and once that has been satisfactorily optimized or some allocated portion of the budget has been expended on observing f_j, we move on to f_{j+1} and so on. Sequential multitask optimization can be thought of as a series of single-objective optimizations where each subsequent optimization problem begins with more and more initial, indirect observations in the form of observations of the previous objective functions. Given the joint GP belief over $\{f_1, f_2, \ldots, f_m\}$, we can directly apply any of the acquisition functions we have previously introduced.

The more interesting formulation is *simultaneous* multitask optimization where any of the m objective functions can be observed in each iteration. Formally, the action space is similar to what we had in multifidelity optimization: $\mathcal{A} = \{1, 2, \ldots, m\} \times \mathcal{X}$. The goal in this setting is to design a utility function that reflects the collective impact of a single observation across all objective functions; one relatively simple option is

$$u'(\mathcal{D}) = \sum_{i=1}^{m} w_i u_i'(\mathcal{D}).$$

The corresponding acquisition function would then be

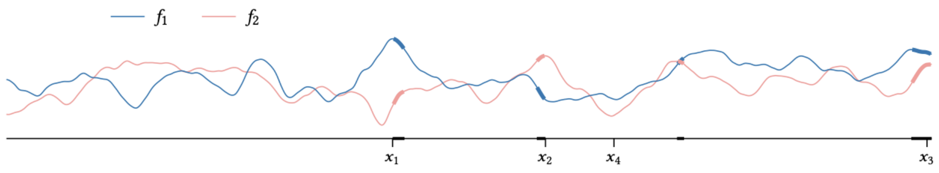
$$a_{\text{MT}}\left((j, \mathbf{x})\right) = \mathbb{E}\left[\sum_{i=1}^{m} w_i u_i\left((j, \mathbf{x})\right)\right] = \sum_{i=1}^{m} w_i \left(\mathbb{E}\left[u_i'\Big(\mathcal{D} \cup \big(j, \mathbf{x}, f_j(\mathbf{x})\big)\Big)\right] - u_i'(\mathcal{D})\right);$$

crucially, because of the joint GP belief over all of the objective functions, it might be the case (depending on how we have defined the function-wise dataset utilities, u_i') that an observation of one objective function can affect the utility of the others.

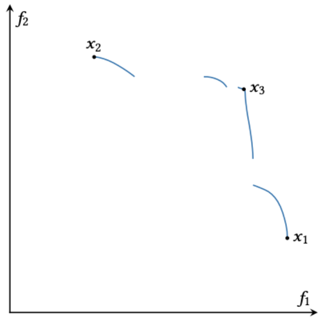
Multiobjective BO

Unlike in the multitask setting, the goal in multiobjective optimization is to find a single point $\mathbf{x}^* \in \mathcal{X}$ that "does well" on all objective functions; of course, unless all objectives are optimized at exactly the same location, we will necessarily need to make tradeoffs between different objective functions.

Figure 8a highlights this need for a simple multiobjective setting for just two objective functions: x_1 maximizes the function f_1 but $f_2(x_1)$ is quite low and similarly, x_2 maximizes f_2 but does not do a good job of optimizing f_1. x_3 corresponds to a location where both functions are relatively high and conversely, x_4 is a pretty poor location for both functions.



(a) A simple multiobjective optimization problem for two (loosely related) objective functions, f_1 and f_2 (top). Four points from the domain, $\{x_1, x_2, x_3, x_4\}$, are highlighted for discussion. The bolded regions along the input axis are all Pareto optimal.



(b) The Pareto frontier associated with the multiobjective task depicted above.

A simple way of assessing the quality of a location in the presence of trade-offs is to check if it is "dominated" by any other location. Formally, we will say that $\mathbf{x}$ dominates $\mathbf{x}'$ if

$$f_j(\mathbf{x}) \geq f_j(\mathbf{x}') \,\forall\, j \in \{1, \dots, m\}.$$

A location is *Pareto optimal* if it is not dominated by any other location. Figure 8a shows the set of all Pareto optimal values in bold. The corresponding function values can be plotted to visualize the *Pareto frontier*, which is shown in Figure 8b. This figure naturally motivates an acquisition function for this setting: observe the location that "pushes out" the Pareto frontier the most i.e., increases the size of the region underneath the Pareto frontier. One nuance is that we need to define an arbitrary, suboptimal reference point to serve as the baseline for this region.

Optimizing this quantity gives rise to the *expected hypervolume improvement* (EHVI) acquisition function; Figure 9 shows an example of EHVI. Unfortunately, computing and optimizing this acquisition function is quite involved and scales poorly with respect to the number of objective functions; researchers have developed efficient approximations which have found varying degrees of empirical success in different settings.

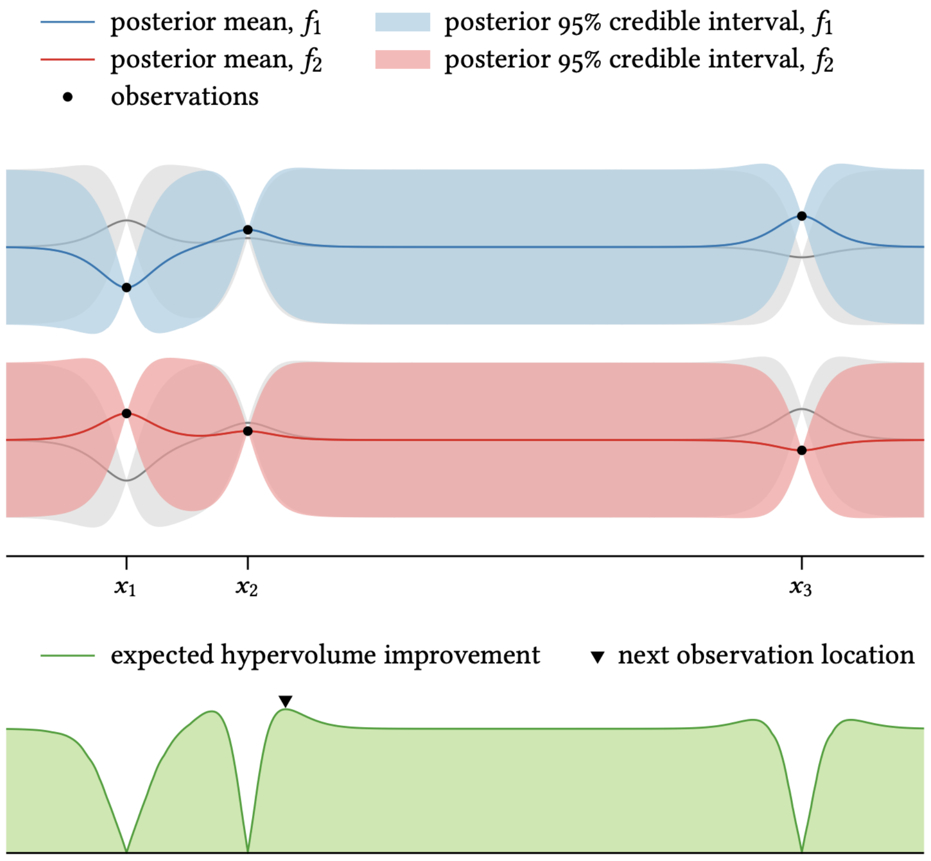


Figure 9: The expected hypervolume improvement acquisition function for a simple two-function multiobjective optimization problem; the two objective functions in this setting are assumed to be independent of each other. The gray posteriors superimposed over each function represent the belief on the other objective.