Optimization Subject to Hidden Constraints via Statistical Emulation

Herbert K.H. Lee
Applied Math & Statistics
University of California, Santa Cruz
herbie@ams.ucsc.edu

Robert B. Gramacy
Booth School of Business
University of Chicago*
rbgramacy@chicagobooth.edu

Crystal Linkletter
Center for Statistical Sciences
Brown University
cdlinkle@stat.brown.edu

Genetha A. Gray
Predictive Simulation R&D
Sandia National Laboratories
gagray@sandia.gov

March 13, 2011

Abstract

We present new methodology for constrained optimization based on building a combination of models, one for the objective function and one for the constraint region. We use a treed Gaussian process as a statistical emulator for the complex objective function, and a random forest to model the probability of meeting the constraints. By combining these models, we can guide the optimization search to promising areas in terms of both the objective function and the constraint. This approach avoids the problem of becoming stuck in a local mode, as well as being able to deal with unconnected viable regions. We demonstrate our methodology on a simulated problem and an example from hydrology.

Key words: constrained optimization, surrogate model, Gaussian process, sequential design, expected improvement

Mathematics Subject Classification: 62M30 62C10 49M30 90C56

*Much of this work was done while R. B. G. was at the Statistical Laboratory, University of Cambridge

1 Introduction

Optimization of complex functions, where many separated local optima abound, is a challenging problem with a long history. Further complication arises when the function can only be successfully evaluated over a subset of the input space, and this region is not known in advance. In this case, it might also be of interest to learn about the feasibility region itself. Our motivating example comes from hydrology, where we are using a computer simulation for designing the placement of a set of wells. We want to minimize the cost of the setup. However, for most runs of the simulator, the response is that there has been a constraint violation, and no further information about cost is provided. Thus we need to simultaneously learn about the constraint region and solve the minimization. A further complication is that no derivative information may be available, so we are restricted to derivative-free optimization methods.

We note that constraints come in several kinds. When the constraints are known restrictions on the valid set of inputs, it is reasonably straightforward to focus the optimization on only the valid set. However when the constraints depend on an evaluation of the function, either on values of the outputs or a failure during the evaluation, the problem becomes considerably more difficult (for example [7] and the references therein). We further distinguish between physical or hidden constraints, where it is not possible to get an output value, and policy or unknown constraints, where an output value is obtained and then deemed not valid. In this paper we focus on the former case, where the computer simulation fails to return a value for some runs. In that case, no information about the objective function can be learned from a run outside the valid set, and one wants to learn the boundaries of valid regions to avoid wasting runs on input settings outside that set.

Our approach is to use statistical emulation, where we build a statistical model as an approximation to the complex simulator, and use this model to guide the optimization. This model has two parts: one to predict the response surface when the simulator gives a valid response, and one to predict whether or not the simulator will return a valid response at all. We use treed Gaussian processes [12] for response surface prediction, and random forests [1] for constraint violation prediction. The former is becoming widely recognized as a flexible surrogate model, or *emulator*, for the design and analysis of computer experiments. The latter is a powerful classification algorithm from the machine learning literature.

In Section 2 we review the statistical emulation techniques. In Section 3 we apply the statistical predictions to optimization using expected improvement. In Section 4 we then provide an illustrative example before finally applying our methods to the motivating example in hydrology in Section 5.

2 Statistical Emulation

The use of stochastic or statistical models to approximate a complex function, such as the output of a computer simulator, is now well established in the literature [26, 20, 27, 6]. In particular, the standard model used for emulation is a Gaussian process (GP). The functional response is treated as a random variable $Z(\mathbf{x})$ that depends on the input vector $\mathbf{x}$ such that the response varies smoothly. The degree of smoothness is determined by the covariance structure of the GP. GPs have the property that any finite set of locations has a joint multivariate normal (Gaussian) distribution, thus the process is completely determined by its mean and covariance function. If we have observations at a set of locations $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, then

$$(Z(\mathbf{x}_1), \ldots, Z(\mathbf{x}_n)) \sim \mathrm{MVN}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{1}$$

where $\boldsymbol{\mu} = (\mu(\mathbf{x}_1), \ldots, \mu(\mathbf{x}_n))$ is a mean function and $\boldsymbol{\Sigma}$ is the variance-covariance matrix. The mean function is typically taken as constant, linear, or a low-order polynomial. The covariance is typically given a simple parameterization such that correlation decreases with distance in the input space. More details on GPs are available in references such as [5] and [28]. We take a fully Bayesian approach, allowing for estimation of uncertainty, which is critical when trying to determine the probability that an unsampled location will be an improvement over the current known optimum.

In some cases standard GP models are adequate. But in others, their limitations, such as strong assumptions of stationarity and poor computational scaling, can be problematic. To reduce these problems, we instead use treed Gaussian process (TGP) models [12]. The basic idea is to use a recursive tree structure to create axis-aligned partitions that segment the space into regions where the assumption of stationarity is valid, and then fit independent GP models within each partition. The approach is an extension of the partition models of [4]. Operating under the Bayesian paradigm, the partitions can be fit simultaneously with the parameters of the embedded GP models using reversible jump Markov chain Monte Carlo [16]. Fitted values and predictions are obtained by averaging over the posterior distribution for the tree structure, so that the model can produce smooth fitted functions. There is software available in the `tgp` library [11] for the open source statistical package R [25].

In addition to emulating the simulator, we also need to predict whether the simulator will return a valid response or whether there will be a constraint violation. For this task we turn to a machine learning tool, random forests. At each training input $\mathbf{x}$, we record a dichotomous variable $Y(\mathbf{x}) = 0$ if the simulator fails to return a valid response, and $Y(\mathbf{x}) = 1$ if a valid response $Z(\mathbf{x})$ is returned (which is used in the construction of the emulator as described above). For example, when n runs are completed, the binary data $Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_n)$ are used to train a collection of B CART trees, each randomly instantiated [2] using m randomly selected splitting variables. We use the average vote over the B trees to obtain the predicted probability that an untried input location $\mathbf{x}$ will return a valid response: $h(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{Y}_b(\mathbf{x})$, where $\hat{Y}_b(\mathbf{x})$ is the classification assigned in the b^{th} randomly started tree. It has been shown that this process converges asymptotically as the number of trees grows large, but that in practice even moderately sized choices for m and B provide a good approximation. More details on random forests are available in [1] and comparison with other classification methods is discussed in [17]. We use the R implementation from the `randomForest` library [22].

3 Optimization

Our approach toward optimization is that of expected improvement (EI) [19, 29]. We use the statistical emulator, or surrogate model, to guide the search for a new optimum, sequentially investigating locations with the largest probability of being an improvement. To be concrete, we focus on the case of minimization (but it can clearly be applied to maximization as well). Herein we assume that the functional response is deterministic, but the method can be generalized for stochastic response functions as well. After N runs, denote the current minimum value observed as $f_{\min} = \min\{z_1, \ldots, z_N\}$, where $z_i = Z(\mathbf{x}_i)$. Define the improvement statistic at a proposed input location $\mathbf{x}$ by

$$I(\mathbf{x}) = \max\{f_{\min} - Z(\mathbf{x}), 0\}. \tag{2}$$

Since the $\mathbf{x}$ of interest is the vector of previously unobserved locations, $Z(\mathbf{x})$ is unknown and we represent its distribution using the posterior predictive distribution from the treed Gaussian process emulator. A simple algorithm for optimization sequentially selects the points $\mathbf{x}'$ that maximize the expected improvement (EI)

$$\mathbf{x}' = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathbb{E}\{I(\mathbf{x})\}. \tag{3}$$

The EI provides a combined measure of how promising a candidate point is, balancing points where the emulator predicts a minimal response with those points where the uncertainty in the emulator is large. For the latter locations, there is a probability of a minimal response even though the mean prediction may not be as small due to the emulator uncertainty.

Our approach for evaluating the EI is with our TGP emulator. As with most problems in Bayesian statistics, we use Markov chain Monte Carlo (MCMC) [10] to fit the emulator, obtaining a Monte Carlo sample of treed Gaussian processes from the posterior distribution. For each of these posterior samples, the expectation calculation (3) is available in closed form as a function of the mean and variance of the (Gaussian) posterior predictive distribution of $Z(\mathbf{x})$. We then obtain a Monte Carlo estimate of the posterior EI by taking an average of the corresponding EI calculations that arise from each sample. Our implementation takes advantage of the functionality of the `tgp` R library, which provides an argument for evaluating the expected improvement, essentially automating the EI calculation.

In the case where there are no constraints, one can just iteratively evaluate the next point that maximizes the EI (as estimated from the emulator), and then update the statistical emulator and the estimated improvement function. There are several algorithms that may be used to aid in the search of the $\mathbf{x}$ input that maximizes the EI. In the case of a simple (non-treed) GP surrogate model and maximum likelihood inference, there is a branch and bound algorithm that may be used [19]. This leads to the so-called expected global optimization (EGO) algorithm. The more sophisticated and fully Bayesian TGP model requires a more sophisticated search method. One useful construct is a Latin hypercube design (LHD) [24] which is a space-filling design of n points that divides the p-dimensional input space into an $n \times n \times \ldots \times n = n^p$ grid and then arranges to have exactly one sample in each row in each dimension. [29] used the TGP EI calculations on random LHDs of candidates as an "oracle" sub-routine within a direct/pattern-search optimizer called APPS [15, 21] and illustrate how these tools may be used to obtain the optimal design for a circuit device. Gramacy and Taddy [13] (Section 3) propose a simpler, R-centric variant via the opposite embedding—where the LHD candidates are augmented by an oracle point obtained by searching the maximum *a posteriori* TGP surface for minima via the `optim` function—that has been shown to perform well in many examples.

In any case, the presence of constraints complicates the notion of expected improvement. It does not make any sense to knowingly waste time evaluating the function in a region expected to return a constraint violation. Thus we need to trade off between the EI and the probability that a valid response will be returned. To do this, we simply multiply the EI by the probability that we will get a valid response. At each tried location, we can record whether or not a valid response was returned by the simulator. This information can be used to inform a random forest classifier. Let $h(\mathbf{x})$ be the predicted probability of a valid response at input $\mathbf{x}$ estimated by the random forest classifier. We extend the algorithm in Equation (3) by now choosing the point $\mathbf{x}'$ that maximizes the expected constrained improvement:

$$\mathbf{x}' = \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbb{E}\{I(\mathbf{x})\} h(\mathbf{x}). \tag{4}$$

With this modification, we can focus our efforts where they are most likely to be fruitful.

How much (4) differs from (3) will depend on the shape of the constraint boundary. Note that this approach takes a global view of the optimization problem, as both the function emulator and the constraint probability map cover the full input space. As with vanilla EI, there are several ways one can proceed to search for the $\mathbf{x}$ which maximizes the EI. All of the techniques mentioned above would apply in the expected constrained improvement context. Our approach is based on the R implementation described by [13].

In summary, an outline of our implementation of this algorithm is:

1. Generate a small Latin Hypercube Design of points and evaluate the function at those points, determining the validity of responses
2. Repeat the following until convergence or computing budget exhausted:
   (a) Fit the TGP emulator and random forest classifier
   (b) Compute the EI and the expected constrained improvement (4) over a set of candidate locations (such as from a LHD)
   (c) Find the candidate which maximizes the expected constrained improvement
   (d) Evaluate this point and determine if its response is valid, and add it to the set of evaluated points
   (e) If the response is valid, move to the next iteration. Otherwise, continue to find the candidate with the next highest expected constrained improvement and evaluate it until a valid point is found, adding all points to the training set for the random forest classifier

For convergence we can monitor the magnitude of the maximum expected constrained improvement and terminate when it is smaller than a threshold value. In practice, we are often faced with a limited computing budget for evaluating the complex objective function, and may need to simply terminate the search when our computing budget is exhausted.

4 Illustrative Example

We provide a relatively simple example here to illustrate our methods. Suppose we want to minimize the function

$$\begin{aligned} Z(x_1, x_2) &= -w(x_1)w(x_2) && (5) \\ w(x) &= \exp\left(-(x-1)2\right) + \exp\left(-0.8(x+1)2\right) - 0.05\sin\left(8(x+0.1)\right). && (6) \end{aligned}$$

Figure 1 shows the negative of the surface (i.e., as a maximization problem) for visibility (it is much easier to see the peaks than the valleys). There are four separated modal regions, and local modes within each of the regions. The true global minimum is shown at $x_1 = -1.0408, x_2 = -1.0408$. Now suppose that when we try to evaluate this function, it will only return valid values inside of an ellipse, but that we don't know anything about this valid region in advance and need to learn about it from the data.

To begin our search, we generate an initial design of 20 random points from a LHD and evaluate the function at those points. We use this initial dataset to fit the treed Gaussian process emulator and the random forest classifier. We then compute the EI at a fresh set of 100 LHD locations, multiply it by the predicted probability of obtaining a valid response at those candidate locations, and choose the maximum of that result as the next point to evaluate. If we obtain a valid response, that completes the iteration. Otherwise, we sample at the location with the next highest expected constrained improvement until we obtain a valid response (note that until a valid response is obtained, there is no need to update

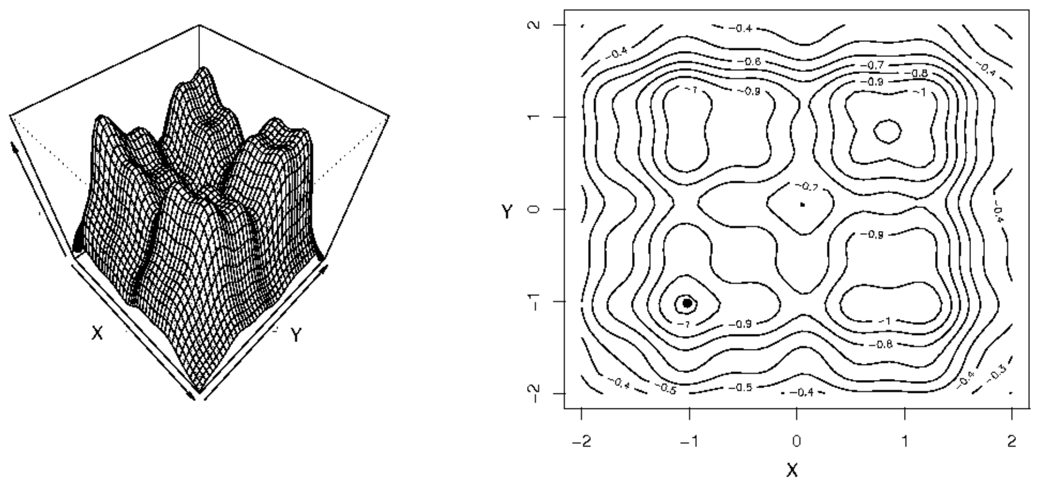

Figure 1: The function to be optimized, shown without constraints and shown as a maximization problem for visibility. The optimum is shown with the dot on the right hand plot.

the emulator). We only retrain the random forests classifier and TGP emulator once per iteration, i.e., only after a valid response is obtained. We do fifty iterations, which provides us with the answer.

Figures 2 and 3 show the results from this implementation. Figure 2 shows an intermediate stage after twenty iterations. The upper left plot shows the truth along with the bounding ellipse (only points inside the ellipse return a valid value) which we are treating as unknown and arbitrarily-shaped. The lower left plot shows the current emulated surface, which is a reasonable approximation of the truth within the valid region, although it will continue to improve as more locations are sampled (and no data is available outside the valid region, so we cannot expect a perfect match outside that region). The solid dots show the locations that have been evaluated, and the open circles show the LHD over which we conduct our search for the next point with highest expected constrained improvement. The top middle plot shows the EI surface. The bottom middle plot shows the predicted probability of returning a valid value obtained from fitting the random forest classifier. The open circles are the locations which have been evaluated successfully, and the filled circles are the locations which have been evaluated but produced a constraint violation. The upper right plot shows the product of the EI and the probability of being valid, and it is this surface that is maximized. The location shown at the crosshairs is the chosen point. The bottom right plot shows the progress in minimization. As the process proceeds, the emulator and the classifier better learn about their respective problems and seek out lower minima until they cannot find anything better. Figure 3 shows the result after thirty iterations (67 function calls). The minimum was found after 39 function evaluations in this run.

For comparison, we pit our approach against a genetic algorithm (GA). The GA was picked as it is another global approach which does not depend on having a good starting location, whereas most local methods require a good staring point with which to initialize the algorithm. We ran our procedure 100 times for 100 iterations each, and we also ran a GA implementation in R [30] 100 times with a population size of 10 for 50 iterations, for a total of 500 function evaluations. On average, our approach required 137.2 function evaluations, and found an average minimum value of -1.0904, whereas the GA was budgeted 500 function evaluations for each run but only found an average minimum value of -1.0864. Our method was within .005 of the true minimum 84% of the time, while the GA was within .005 only 58% of the time, despite using many more function evaluations.

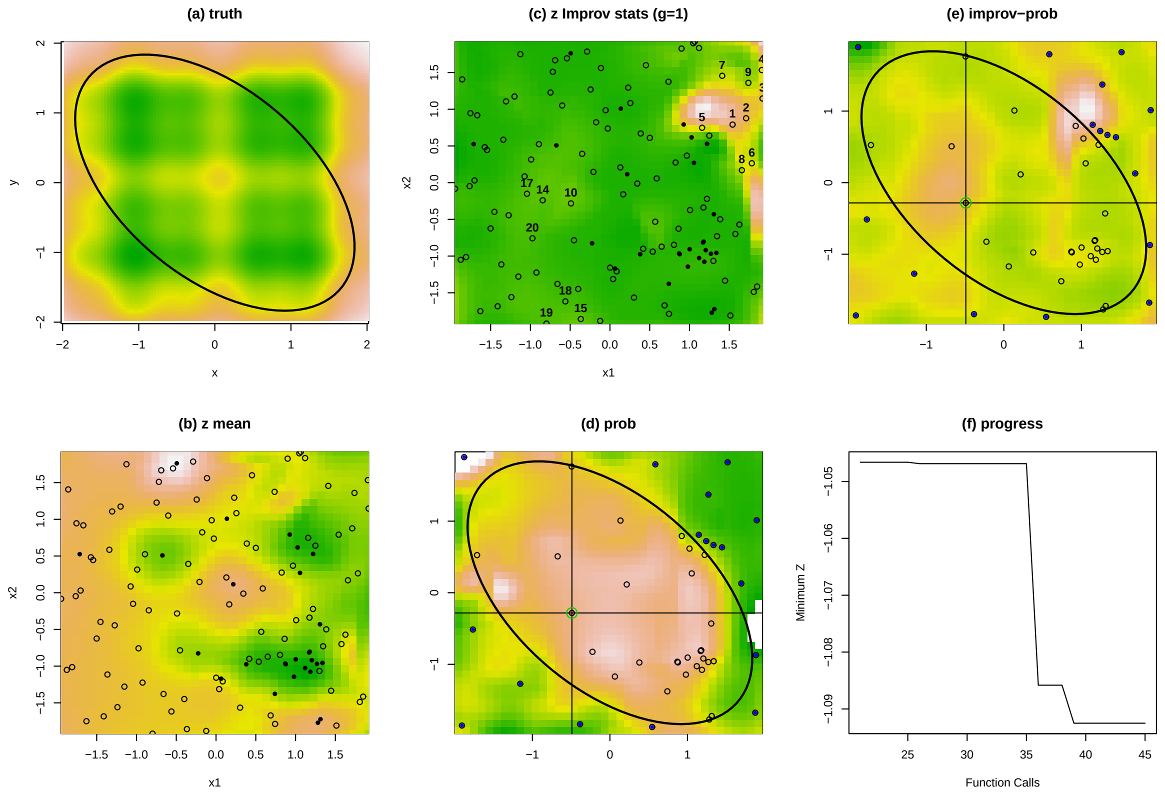

Figure 2: After 20 iterations: (a) the function to be optimized with valid region, (b) the fitted emulator, (c) the EI surface, (d) the fitted probability of being valid, (e) the expected constrained improvement, and (f) the progress in minimization. White shows high values and green (dark) shows low values.

5 Hydrology Example

A key motivating example for this work is the hydraulic capture problem from the community problems [23]. The goal of the problem is to find a configuration of up to four wells to control the direction of groundwater movement to contain a contaminant plume, and to do so at minimal cost. The constraints in this problem are in the form of specifications on the hydraulic head differences at designated locations, and the objective function is the monetary cost of installing and operating the wells. More details on this problem are available in [23]

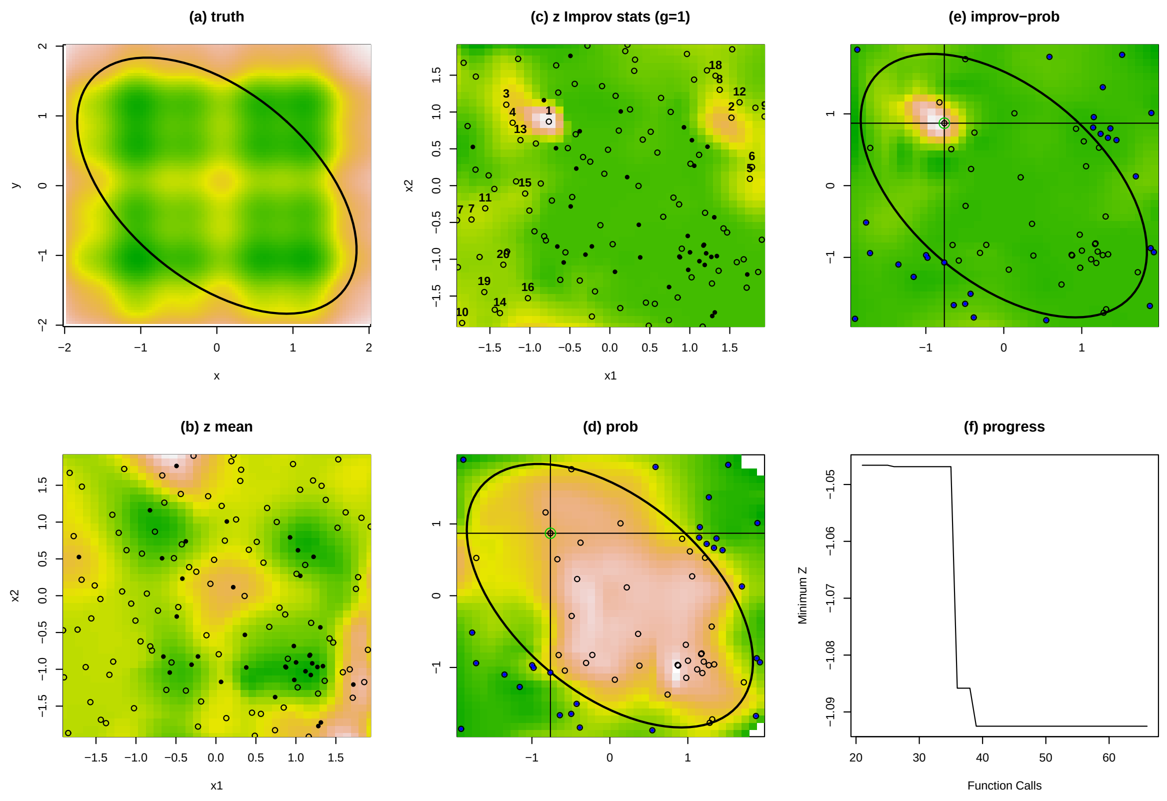

Figure 3: After 30 iterations: (a) the function to be optimized with valid region, (b) the fitted emulator, (c) the EI surface, (d) the fitted probability of being valid, (e) the expected constrained improvement, and (f) the progress in minimization. White shows high values and green (dark) shows low values.

and [8]. A number of solutions have been proposed for this problem, and some comparisons and discussion of the difficulties are available in references such as [9, 14, 18]. This problem is particularly challenging for two reasons. First, the objective function is quite irregular and difficult to predict. Second, the dimension of the space is not fixed, varying from twelve dimensions if four wells are used, down to three dimensions if only one well is used (the three parameters corresponding to each well are detailed below).

The hydraulic capture problem is described in [23] and the implementation details can be found in [9]. Intuitively, the problem is illustrated in Figure 4. The green oval shows the relative location of a contaminant plume in a rectangular aquifer. The circles around the exterior of the plume represent the gradient constraints on the hydraulic head values, aligned so that flow will be forced towards the interior of the plume to prevent migration. Groundwater flow in the natural system in the absence of wells is towards the northeast, indicated by the arrow in the lower left corner. In this illustration, we also use stars to show a reasonable configuration of four wells—two extracting water inside the plume and two injecting water outside the plume. The inputs of the objective function are the well locations (in x-y coordinates) and the flow rates of each of the wells, so that there are three parameters corresponding to each well. The function output is the total cost of installing and operating the wells. Note that the calculation of this cost requires the results from the groundwater simulator.

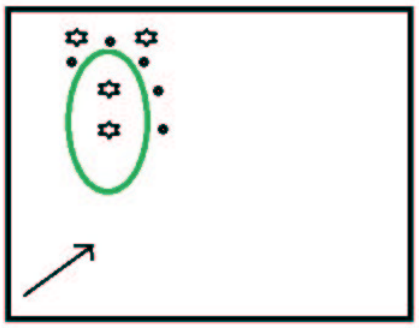

Figure 4: Pictorial representation of the constrained optimization problem

Our first approach is to start with four wells and allow the search to work its way down to a single well iteratively. When the flow rate of a well is less than 1×10^{-4}, the well is turned off and the flow rate is set to zero. The search is initialized by seeding the space with a LHD to obtain some points to fit the emulator. We then propose a set of candidate points from a fresh LHD, but we augment the candidate set over which we evaluate the expected improvement in two ways. First, following [29], we add fifty points in a ball of small radius around the best point, which helps in exploring higher-dimensional spaces. Second, we add a candidate point equal to the best point with the smallest well rate set to flow zero. If this point is determined to be the best, it allows us to remove a well from the design and move to a lower-dimensional problem. We then fit the TGP emulator and the random forest classifier, and rank the candidate points based on the product of their expected improvement and probability of being valid, as per Equation (4). We evaluate the points in rank order until one returns a valid output, then we move to the next iteration, updating the statistical models and repeating. Convergence can be assessed by tracking the maximum expected constrained improvement. This algorithm is able to easily reduce the number of wells from four to two, but it has more difficulty moving to a single well solution. The best solution it finds is a cost of \$36,389.83 using two wells. We note that this solution is better than the solutions found by six of the nine methods used in a comparison study of techniques on this problem by [9].

However, this two well solution is not the best solution. We take a second approach to searching the space, using our expected improvement approach for each of one-, two-, three-, and four-well spaces separately. In this case, we do not supplement the candidate search with a point with fewer wells, nor do we remove wells with extremely small flow rates. With this fixed-dimension approach, we find a best one-well solution of \$22,952.77, which is better than all of the solutions found in [9], with the best solution reported there being \$23,421. We note that the setup of the experiment here is different than in the alternatives; our method requires more than one initial valid point to initialize the statistical models. The comparison methods might perform differently under the same initial conditions we used here. For a small number of possible wells, it is more efficient for us to search each space separately, as the statistical models can train more effectively to each state space, but this approach would not scale as well to larger dimensional spaces.

6 Conclusions

In conclusion, we have proposed a new approach for constrained optimization based on statistical models. By simultaneously learning about both the objective function and the boundary of the constraints, we are able to better focus our efforts on the valid regions. We note that because our methods are based on statistical models, they also apply to the cases of stochastic simulators or probabilistic constraints.

Some areas for further work include the consideration of other classification models. For example, when the constraint boundary is relatively smooth, a Gaussian process classifier [3] may learn it more quickly. It would be worthwhile to compare the relative efficiency of different classifiers under different types of feasible regions (e.g., connected versus disjoint feasible regions). Another possibility is to connect the surrogate model with the model for estimating the constraint boundary. A different approach is to move beyond expected improvement at a point and consider a more global improvement function, such as one that integrates improvement over the whole space. Convergence diagnostics could also be better explored, such as those along the lines of [13] (Section 3).

Acknowledgments

This research was initiated at a workshop at the American Institute of Mathematics on Derivative-Free Hybrid Optimization Methods for Solving Simulation-Based Problems in Hydrology, and was also partially supported by NSF grant DMS-0906720 to HKHL and EPSRC grant EP/D065704/1 to RBG. We thank the reviewer and associate editor for their helpful comments.

References

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

[2] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[3] T. Broderick and R. B. Gramacy. Classification and categorical inputs with treed gaussian process models. *Journal of Classification*, 2011. to appear.

[4] H.A. Chipman, E.I. George, and R.E. McCulloch. Bayesian treed models. *Machine Learning*, 48:303–324, 2002.

[5] N. A. C. Cressie. *Statistics for Spatial Data, revised edition.* John Wiley & Sons, 1993.

[6] Kai-Tai Fang, Runze Li, and Agus Sudjianto. *Design and Modeling for Computer Experiments.* Chapman & Hall/CRC, Boca Raton, 2006.

[7] D. E. Finkel and C. T. Kelley. Convergence analysis of sampling methods for perturbed lipschitz functions. *Pacific Journal of Optimization*, 5:339–350, 2009.

[8] K. R. Fowler, C. T. Kelley, C. E. Kees, and C. T. Miller. A hydraulic capture application for optimal remediation design. In C. T. Miller, M. W. Farthing, Gray G. W., and G. F. Pinder, editors, *Proceedings of the XV International conference on Computational Methods in Water Resources*, 2004.

[9] K. R. Fowler, J. P. Reese, C. E. Kees, J. E. Dennis Jr., C. T. Kelley, C. T. Miller, C. Audet, A. J. Booker, G. Couture, R. W. Darwin, M. W. Farthing, D. E. Finkel, G. Gablonsky, G. Gray, and T. G. Kolda. A comparison of derivative-free optimization methods for water supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, May 2008.

[10] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall, London, 1995.

[11] Robert B. Gramacy. `tgp`: An `R` package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. *Journal of Statistical Software*, 19(9), 2007.

[12] Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103:1119–1130, 2008.

[13] Robert B. Gramacy and Matthew Alan Taddy. Categorical inputs, sensitivity analysis, optimization and importance tempering with `tgp` version 2, an `r` package for treed gaussian process models. *Journal of Statistical Software*, 33(6):1–48, 02 2010.

[14] G. A. Gray, K. R. Fowler, and J. D. Griffin. A hybrid optimization scheme for solving the hydraulic capture problem with an unknown number of wells. In *The First Interna-*

tional Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering, 2009.

[15] Genetha A. Gray and Tamara G. Kolda. Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software*, 32(3):485–507, 2006.

[16] P.J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82:711–732, 1995.

[17] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer-Verlag, New York, NY, 2001.

[18] T. Hemker, K. R. Fowler, M. W. Farthing, and O. von Stryk. A mixed-integer simulation-based optimization approach with surrogate functions in water resources management. *Opt. Eng.*, 9(4):341–360, 2008.

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

[20] M. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87:1–13, 2000.

[21] Tamara G. Kolda. Revisiting asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal of Optimization*, 16(2):563–586, December 2005.

[22] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[23] A. S. Mayer, C. T. Kelley, and C. T. Miller. Optimal design for problems involving flow and transport phenomena in saturated subsurface systems. *Advances in Water Resources*, 25:1233–1256, 2002.

[24] M. D. McKay, W. J. Conover, and R. J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[25] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[26] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, 1989.

[27] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer-Verlag, New York, NY, 2003.

[28] Michael L. Stein. *Interpolation of Spatial Data*. Springer, New York, NY, 1999.

[29] Matthew Taddy, Herbert K. H. Lee, Genetha A. Gray, and Joshua D. Griffin. Bayesian guided pattern search for robust local optimization. *Technometrics*, 51:389–401, 2009.

[30] Egon Willighagen. *genalg: R Based Genetic Algorithm*, 2005. R package version 0.1.1.