Enhancing Parallel Pattern Search Optimization with a Gaussian Process Oracle

Genetha A. Gray, Monica Martinez-Canales
Computational Sciences & Mathematics Research Department, Sandia National Laboratories, PO Box 969 MS 9159, Livermore, California 94551-0969, USA, {gagray@sandia.gov, mmarti7@sandia.gov}

Herbert K. H. Lee, Matt Taddy
Department of Applied Mathematics & Statistics, University of California, Santa Cruz, {herbie@ams.ucsc.edu, taddy@soe.ucsc.edu}

Robert B. Gramacy
The Statistical Lab, University of Cambridge, {}

We consider a derivative-free method from the pattern search class of algorithms for the solution of simulation-based optimization problems. Because simulations often require significant computational time and resources, we are striving to reduce the number of runs needed by the optimization method. Moreover, since pattern searches are local methods, we are investigating ways of introducing robustness and some global properties. To accomplish these goals, we are using ideas from the design of computer experiments literature and using random functions to model the deterministic computer output function. We treat the output of the simulations as realizations of a Gaussian Process (GP). Then, the uncertainty about future computer evaluations can be quantified by finding the predictive distribution for new input locations conditional on the points that have already been evaluated. These ideas have been adapted to complex computer simulations in an R code referred to as tgp. This work combines the search properties of a pattern search with the statistical properties of the GP to create a new hybrid algorithm. In this paper, we will describe the optimization algorithm, the GP algorithm, and the resulting hybrid method, and we will present some numerical results.

Key words: derivative-free optimization, Gaussian Process, expected improvement, oracle, local convergence

1 Introduction and Motivation

Significant advances in computing capabilities, decreasing storage costs, and the rising costs associated with physical experiments have contributed to an increase in the use of numerical modeling and simulation. In recent years, simulation has frequently been paired with optimization in order to design and control complex engineering systems. The resulting simulation-based optimization problems have an objective function whose evaluation requires the results of a complex simulation.

Surrogate-based optimization has become a common approach for solving simulation-based optimization problems. The essence of such approaches entails constructing low fidelity models by fitting response surfaces to high fidelity function values or by reducing the numerical or physical fidelity of the simulation. Optimization methods are then applied to these less expensive low fidelity functions with periodic corrections from the high fidelity simulation to ensure convergence to a local minimum of the high fidelity function. Many variations of this type of approach can be found in the literature. Examples include [**?**, **?**, **?**, **?**, **?**]. *GENETHA: Add these references to the bib file*

We expand on these ideas and leverage ideas from Bayesian statistics to construct a new optimization under uncertainty (OUU) algorithm. In an attempt to reduce the overall computing expenses associated with simulation-based optimization, we constructed an OUU algorithm using a local optimization method and a "global" Gaussian Process (GP). The optimization method, APPSPACK, is described in Section 2, and the Gaussian Process, tgp, is detailed in Section 3. Section 4 describes the specifics of the new OUU algorithm resulting from the hybridization of the GP and the optimization method. Numerical results are presented in Section 5. The development of such an optimization algorithm, one that combines both global and local information, is an area of future research. Section 6 touches on some related topics for further study.

2 Derivative-Free Optimization

Simulation-based optimization problems have an objective function whose evaluation requires the results of a complex simulation and are often characterized by a relatively small number of variables (i.e., $n < 100$). Moreover, derivatives often do not exist and/or are difficult to estimate. Therefore, derivative-free methods have become a useful method of solution.

2.1 The APPS Algorithm

In this work, we consider a derivative-free optimization method called Asynchronous Parallel Pattern Search (APPS) [6, 7]. The APPS algorithm is part of a class of direct search methods which were primarily developed to address problems in which the derivative of the objective function is unavailable and approximations are unreliable [12, 15]. Pattern searches use a predetermined pattern of points to sample a given function domain. It has been shown that if certain requirements on the form of the points in this pattern are followed and if the objective function is suitably smooth, convergence to a stationary point is guaranteed [2, 11, 14].

The majority of the computational cost of pattern search methods is the function evaluations, so parallel pattern search (PPS) techniques have been developed to reduce the overall computation time. Specifically, PPS exploits the fact that once the points in the search pattern have been defined, the function values at these points can be computed simultaneously [1, 13]. The APPS algorithm is a modification of PPS that eliminates the synchronization requirements. It retains the positive features of PPS, but reduces processor latency and requires less total time than PPS to return results [6]. Implementations of APPS have minimal requirements on the number of processors and do not assume that the amount of time required for an objective function evaluation is constant or that the processors are homogeneous.

We consider the specific APPS algorithm as described in [7]. It is a variant on generating set search as described in [8] and is provably convergent under mild conditions [7, 9, 10]. Omitting the implementation details, the basic APPS algorithm can be simply outlined as follows:

1. Generate a set of trial points to be evaluated.
2. Send the set of trial points to the *conveyor* for evaluation, and collect a set of evaluated points from the conveyor. (The conveyor is a mechanism for shuttling trial points through the process of being evaluated.)
3. Process the set of evaluated points and see if it contains a new *best point*. If there is such a point, then the iteration is successful; otherwise, it is unsuccessful.
4. If the iteration is successful, replace the current best point with the new best point. Optionally, regenerate the set of search directions and delete any pending trial points in the conveyor.
5. If the iteration is unsuccessful, reduce certain step lengths as appropriate. In addition, check for convergence based on the step lengths.

A detailed procedural version of APPS is given in [3], and a complete mathematical description and analysis is available in [7].

2.2 APPSPACK

The APPS algorithm described here has been implemented in an open source software package called APPSPACK. It is written in C++ and uses MPI [4, 5] for parallelism. The details of the implementation are described in detail in [3]. There are both serial and parallel versions of APPSPACK, but to achieve the goals of this work, we are solely interested in the parallel version.

APPSPACK performs function evaluations through system calls to an external executable, and no restrictions are placed on the language of this executable. This simplifies its execution and makes it amenable to customization. Of particular interest to us is the management of the function evaluation process. The procedure is quite general and merely one way of handling the process of parallelizing multiple independent function evaluations and efficiently balancing computational load. This management system makes APPSPACK amenable to the hybridization proposed in Section 4.

APPSPACK has been successfully applied to problems in microfluidics, biology, groundwater, thermal design, and forging. (See [3] and references therein.)

3 Gaussian Processes

Because simulations often require significant computational time and resources, we are striving to reduce the number of runs needed by the optimization methods like APPSPACK. Moreover, since APPSPACK is a local optimization method, we are investigating ways to add robustness and introduce global properties. To accomplish these goals, we are using ideas from the design and analysis of computer experiments literature and using random functions to model the deterministic computer output function.

Following the standard practice in the literature [**?**, **?**, **?**], we start by modeling the output of the simulations as a realization of a Gaussian Process (GP). A GP is a stochastic process for which any finite collection of observations has a multivariate Gaussian distribution [**?**]. The process is described by its mean function $\mu(\cdot)$ and its covariance function $C(\cdot,\cdot)$. For a set of locations $\mathbf{s}_1, \ldots, \mathbf{s}_n \in \mathcal{S}$, we can write the distribution for the observations $\mathbf{x}$ as

$$\mathbf{x} \sim MVN\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

where $\mathbf{x} = (x(\mathbf{s}_1), \ldots, x(\mathbf{s}_n))$, $\boldsymbol{\mu} = (\mu(\mathbf{s}_1), \ldots, \mu(\mathbf{s}_n))$ and $\boldsymbol{\Sigma}$ is the variance-covariance matrix with elements $C(\mathbf{s}_i, \mathbf{s}_j)$. We typically take the mean function to be linear in location, although lower-order polynomials can be used, as can the assumption of a constant mean. If we make the typical assumptions of stationarity (C does not depend on location, but only on the distance between locations) and isotropy (the distance metric is spherically symmetric), we can simplify the expression for the covariance matrix to

$$C(\mathbf{s}_i, \mathbf{s}_j) = \frac{1}{\lambda}\rho(d)\,,$$

where $\rho()$ is the correlation function and d is the Euclidean distance between $\mathbf{s}_i$ and $\mathbf{s}_j$. ρ is then taken to be a parametric family such as the Matérn class [**?**], or the power family, which is what we use here:

$$\rho(d) = \exp\left[-\left(\frac{d}{\theta}\right)^{\gamma}\right]\,.$$

The choice of $\gamma = 1$ gives the exponential correlation function, while $\gamma = 2$ give the Gaussian correlation function which is the one we use hereafter. This correlation function is easily expanded to account for anisotropy by allowing different range parameters θ_i for each dimension.

The model can be fit either by optimization in a maximum likelihood framework, or using Markov chain Monte Carlo in a Bayesian framework. Here we take the Bayesian approach, which allows incorporation of prior information about location or uncertainty in the parameter distributions. This allows full propagation of uncertainty, so that predictive uncertainty can be incorporated into the selection of new potential optimization points as described in Section 4.1. We use standard priors for the parameters, after standardizing the data [**?**]. More information on choices of priors and fitting of the model can be found in [**?**], [**?**], and [**?**].

In practice, many situations, including computer simulations, call for more flexibility than is reasonable under the assumption of stationarity. Here we use treed Gaussian process (TGP) models [**?**] to achieve nonstationarity. This approach partitions the input space and fits separate GPs within each partition. The partitions are fit simultaneously with the individual GP parameters using reversible jump Markov chain Monte Carlo, so that all parts of the model can be learned automatically from the data. The posterior predictive distribution thus takes into account uncertainty from the data, from the fitted parameters, as well as the fitted partitions, and is ideal for use within the oracle process in Section 4.1. Implementation is via the library TGP for the open source statistical package R.
(see `http://www.cran.r-project.org/src/contrib/Descriptions/tgp.html`).

4 APPSPACK-tgp

In recent years, some optimization methods have introduced an *oracle* to predict points at which a decrease in the objective function might be observed. These points are given in addition to the iterates inherent to the optimization method itself. Analytically, an oracle is free to choose points by any finite process. (See [8] and references therein.) The addition of an oracle is particularly amenable to a pattern search methods like APPS. The iterate(s) suggested by the oracle are merely additions to the pattern. Furthermore, the asynchronous nature of the APPSPACK implementation makes it adept at handling the evaluation of the additional points. In fact, the oracle structure was successfully used in the design of a multifidelity optimization method [?, ?].

In this work, we use the tgp statistical model as our oracle in the hopes of adding robustness and introducing some global properties to APPSPACK. When the oracle is called, the tgp algorithm is applied to the set of evaluated iterates in order to choose additional candidate points. In other words, APPSPACK is still optimizing as normal, but throughout the optimization process, the iterate pairs $(x_i, f(x_i))$ are collected. Then, the tgp algorithm considers the current set of iterate pairs and recommends one or more new iterates. These points are then evaluated to see if one of them is a new best point. If not, the point is merely discarded. However, if a tgp point is a new best point, the APPSPACK search pattern continues from its location.

4.1 Expected Improvement Criteria

As mentioned above, the oracle treats the output of the simulations as realizations of a Treed Gaussian Process. The uncertainty about future computer evaluations can be quantified by finding the predictive distribution for new input locations conditional on the points that have already been evaluated. Since we now have a full probabilistic model for code output at new locations, any statistic depending upon this output is easily obtained.

The expected improvement at a point x, $\mathbb{E}[\min(f_{min} - f(x), 0)]$, is a useful criteria for choosing new locations for evaluation. The paper by Jones et al [?] illustrates the use of this statistic in optimization. Since the improvement is a random variable, this criteria balances rewarding points where the output is highly uncertain, as well as where the function is generally predicted to be better than the present best point. Each time that the oracle is called, a number of candidate locations are generated from an optimal space filling design. The TGP model is then fit to the existing output, and the expected improvement is calculated at each candidate location. The points with highest expected improvement are passed back to the APPS algorithm for evaluation.

4.2 Theoretical Considerations

One of the main theoretical considerations in developing a hybrid method like the one described here is convergence. In this case, the APPS algorithm is provably convergent under mild conditions. This result can be leveraged since we incorporate the iterate(s) suggested by the Gaussian process as an oracle. Since the oracle points are given in addition to those generated by the pattern search, there is no adverse affect on the convergence.

Future work includes investigating improvement to the convergence as a result of the tgp oracle. Moreover, we hope to incorporate the findings of tgp into the APPSPACK stopping criterion. Currently, the primary stopping condition is based on the step length. This criterion was chosen because it can be shown that if the objective function is continuously differentiable, then the norm of the gradient (or an analogous measure of the constrained measure of stationarity in the bound-constrained case) can be bounded as multiple of the step size [8]. In other words, the steps only get smaller if the norm of the gradient is decreasing. Alternatively, APPSPACK offers two additional stopping criterion. One is based on whether or not the function has reached a specified threshold, and the other is defined in terms of the number of function evaluations. We would like to add the additional stopping condition that depends upon the expected improvement over the entire input space, a statistic that is calculated each time the oracle is called. This will ensure that the tgp oracle does not stop if there is high probability of finding a better location.

5 Numerical Results

In practice, we have implemented our ideas and tested them with some promising results. Here, we share our results for the two dimensional fine resolution test problem shown in Figure 5. The equation for this

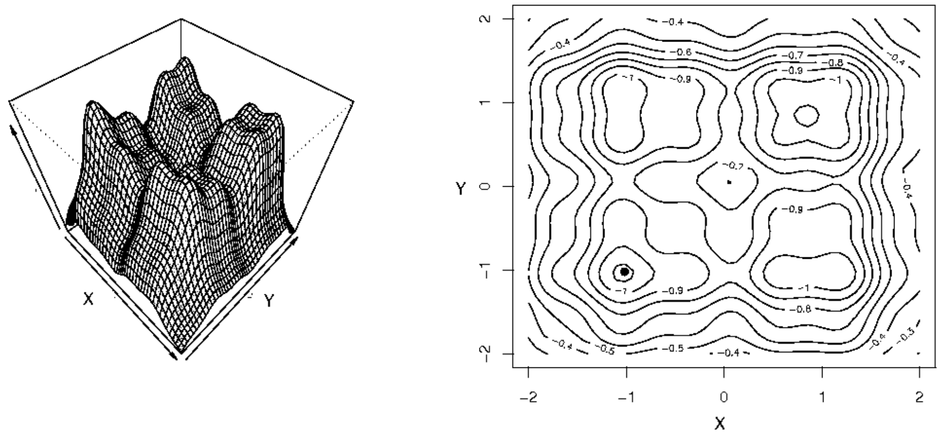

Figure 1: Plot of the function used for the numerical testing. The left shows the function while the left shows its contour lines. The true maximum, at $x = -1.04$ $y = -1.04$, is marked on the plot.

function is

$$\begin{aligned} f(x, y) \quad &= \quad \left[-\exp\left(-(x-1)^2\right) - \exp\left(-0.8(x+1)^2\right) + 0.05\sin\left(8(x+0.1)\right)\right] \\ &\quad * \left[\exp\left(-(y-1)^2\right) + \exp\left(-0.8(y+1)^2\right) - 0.05\sin\left(8(y+0.1)\right)\right] \end{aligned}$$

As shown in Figure 5, the oracle enhanced version of APPSPACK converges to $f = 1.126$ at $x = [-1.042, -1.058]$ which is the correct answer. Finding this solution took 63 total function evaluations, where 15 resulted from points suggested by the tgp oracle.

Moreover, using the same initial values, APPSPACK without the tgp oracle completes 41 function evaluations and converges to a local maximum $x = [1.125, 0.632]$ where $f = 1.043$. Examination of the expected improvement statistic shows that it is close to zero everywhere in the case of APPSPACK-tgp. However, in the case of APPSPACK alone, large variations in the expected improvement were observed over the input space suggesting that the tgp oracle would have improved the optimization method by suggesting iterates from different parts of the input space. This is illustrated in 5. At the time of convergence, large variations in the expected improvement were observed over the input space (see Figure 5), suggesting that the tgp oracle would have improved the optimization method by including iterates from different parts of the input space.

Indeed, the maxima found with the oracle enhanced version of APPSPACK were always higher than the solution from APPSPACK without an Oracle. However, there is currently no guarantee of locating the global solution if APPSPACK converges before the input space is fully explored. When the algorithm converges to the global maximum, as shown in Figure 5, we see that the expected improvement is everywhere zero. But after convergence to a local maximum, there is still positive expected improvement over the input space, as illustrated in Figure 5. In Figure 5, we show an example where APPSPACK-tgp converges to a local maximum $f = 1.061$ at $x = [1.125, 1.137]$ in 42 total function evaluations, where 8 iterates resulted from the tgp oracle. This observation warrants investigation into a convergence criteria that includes the expected improvement information provided by the oracle. Regardless, the tgp oracle allows us to make an intelligent assessment of the robustness of our solution and to obtain a probabilistic view of the global output surface.

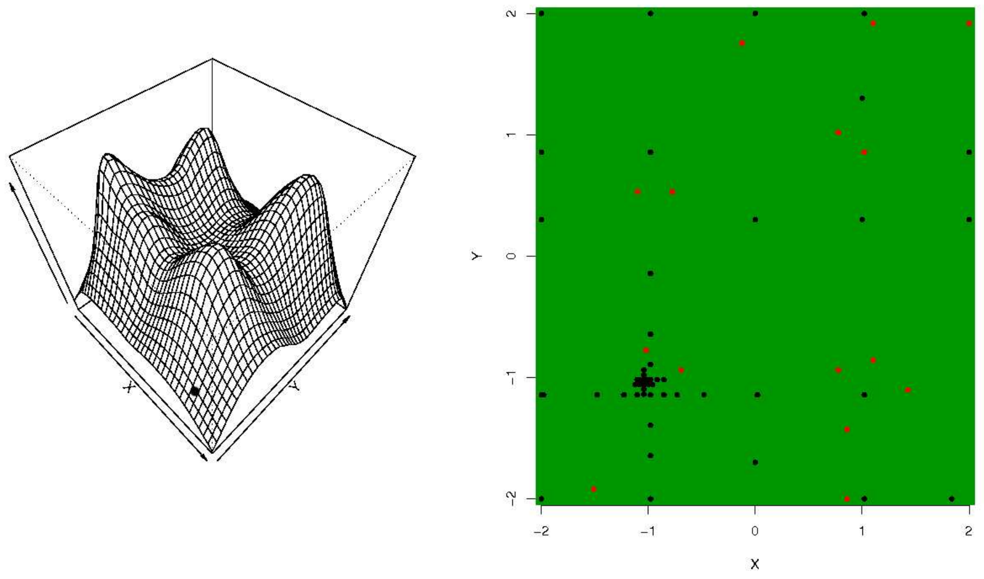

Figure 2: The mean predicted output surface is illustrated on the left and expected improvement over the input space is shown on the right for the APPSPACK-tgp hybrid method. In the expected improvement picture, the black dots are evaluated iterates chosen by APPSPACK while the red dots are evaluated iterates chosen by the tgp oracle.

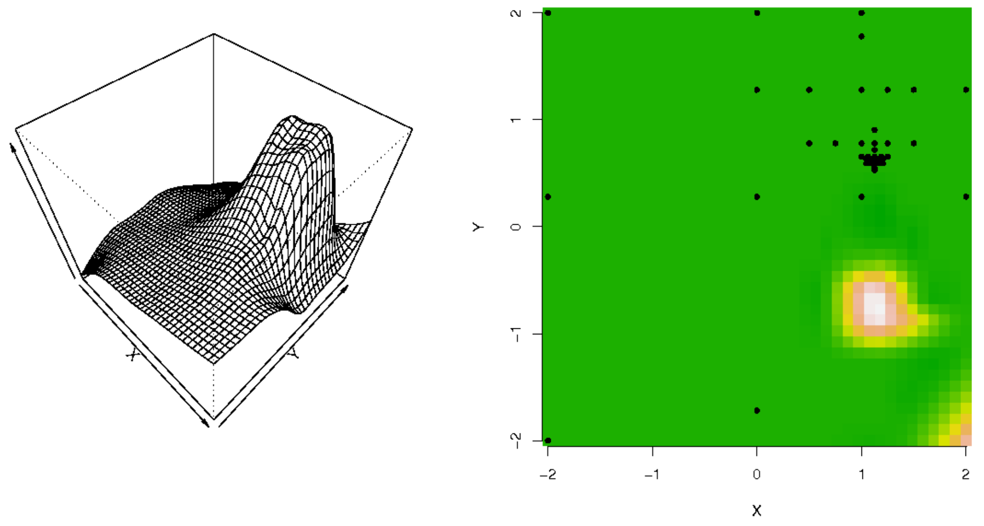

Figure 3: The mean predicted output surface (left) and the expected improvement over the input space (right) for the results of APPSPACK without the oracle.. The black dots are the points evaluated during the pattern search.

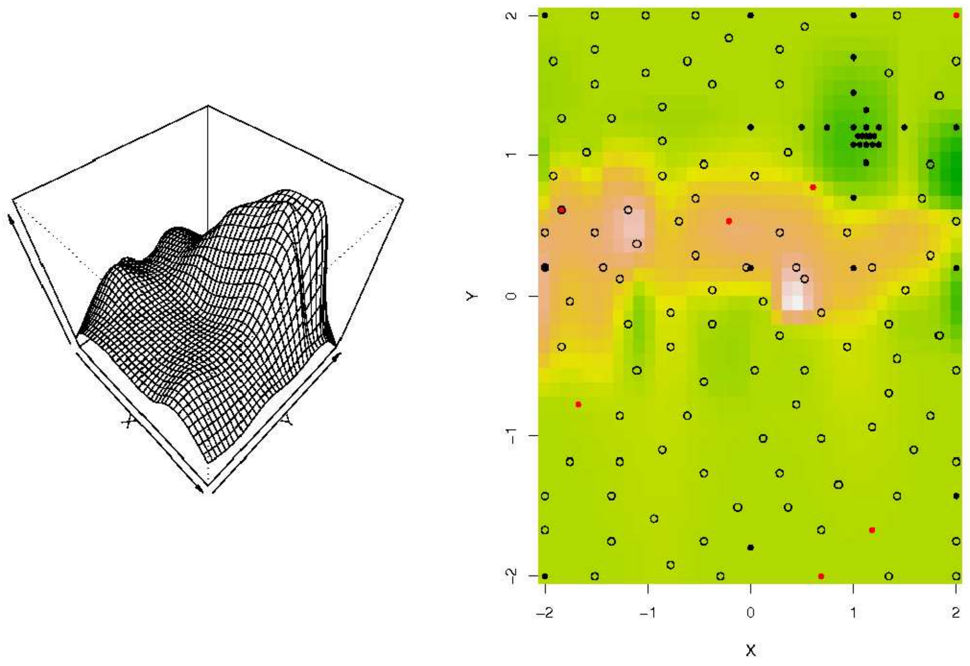

Figure 4: The mean predicted output surface (left) and the expected improvement over the input space (right) for APPSPACK-tgp. On the right, the black dots are evaluated points chosen by APPSPACK, the red dots are evaluated points chosen by the tgp oracle, and the open circles are candidate locations from a treed D-optimal design.

6 Discussion and Future Work

This algorithm represents some of ideas that need to be incorporated if optimization algorithms are to be used as decision makers. Current optimization algorithms accept guesses (or iterates) based solely on some notion of (sufficient) increase or decrease of the objective function. In order to serve as decision makers, next-generation algorithms must also consider rankings and probability metrics. For example, computational costs can be assessed so that iterates are only evaluated if they meet a set computational budget. This is particularly important for the expensive objective functions of simulation-based optimization problems. Moreover, hybridization optimization algorithms such as the one suggested here will allow the user to dismiss subsets of the domain that exhibit large variances or that exceed critical thresholds.

In addition, this algorithm is a step towards finding methods capable of generating a “robust” set containing multiple optima instead of a single opimum. The current state of the art is to accept an iterate as an optimum based on the inability to find better guess within a decreasing search region. This may lead to solutions to design problems that are undesirable due to a lack of robustness to small design perturbations. Our goal is to introduce algorithms that allow designers to choose a solution based on additional criteria. For example, a regional optimum could be used to generate a set of multiple solutions from which the designer can choose.

7 Acknowledgments

This project thanks the Sandia National Laboratories Laboratory Directed Research and Development (LDRD) program, and the Computational and Information Sciences (C&IS) and Engineering Sciences (ES) Investment Areas, in particular, for their funding support.

References

[1] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM J. Opt.*, 1:448–474, 1991.

[2] E. D. Dolan, R. M. Lewis, and V. Torczon. On the local convergence properties of parallel pattern search. Technical Report 2000-36, NASA Langley Research Center, Institute for Computer Applications in Science and Engineering, Hampton, VA, 2000.

[3] G. A. Gray and T. G. Kolda. APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. Technical Report SAND2004-6391, Sandia National Laboratories, Livermore, CA 94551, August 2004.

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comp.*, 22:789–828, 1996.

[5] W. D. Gropp and E. Lusk. User’s guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[6] P. D. Hough, T. G. Kolda, and V. Torczon. Asynchrounous parallel pattern search for nonlinear optimization. *SIAM J. Sci. Comput.*, 23:134–156, 2001.

[7] T. G. Kolda. Revisiting asynchronous parallel pattern search. Technical Report SAND2004-8055, Sandia National Laboratories, Livermore, CA 94551, February 2004.

[8] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[9] T. G. Kolda and V. Torczon. Understanding asynchronous parallel pattern search. In Gianni Di Pillo and Almerico Murli, editors, *High Performance Algorithms and Software for Nonlinear Optimization*, volume 82 of *Applied Optimization*, pages 316–335. Kluwer Academic Publishers, Boston, 2003.

[10] T. G. Kolda and V. Torczon. On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization*, 14(4):939–964, 2004.

[11] R. M. Lewis and V. Torczon. Rank ordering and positive basis in pattern search algorithms. Technical Report 96-71, NASA Langley Research Center, Institute for Computer Applications in Science and Engineering, Hampton, VA, 1996.

[12] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191–207, December 2000.

[13] V. Torczon. PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines. Technical Report TR92-09, Rice University, Department of Computational & Applied Math, Houston, TX, 1992.

[14] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Opt.*, 7:1–25, 1997.

[15] M. H. Wright. Direct search methods: Once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, volume 344 of *Pitman Research Notes in Mathematics*, pages 191–208. CRC Press, 1996.