Parallelized Direct Search of a Binary Objective

P. Grant
Department of Electrical Engineering,
Vanderbilt University,
Nashville, TN 37235

D. G. Walker
Thermal Engineering Laboratory,
Department of Mechanical Engineering,
Vanderbilt University,
Nashville, TN 37235
e-mail: greg.walker@vanderbilt.edu

Many scientific and engineering problems are solved by utilizing simulations of computationally intensive mathematical models within massive design spaces. As a result, parametric studies of these models are cost prohibitive in terms of computational time. The focus of this work is a particular kind of parameter sweep problem where the simulation of the model, given real parameters, results in a binary value. The goal of this work is to design, develop, and implement a parallel algorithm for bounding a binary objective when the simulation is a computationally intensive mathematical model. A fully functioning implementation is provided for a two-dimensional example using client-server architecture. Results show that a straight bisection search is approximately 50% faster than a full parametric sweep for most continuous functions. With parallelization and load balancing, the simulation is remarkably faster, exhibiting near-linear speedup up to 16 processors for most functions. [DOI: 10.1115/1.3510589]

1 Introduction

Engineering analysis is often concerned with identifying conditions responsible for catastrophic failure. For example, consider the load required to buckle a column. Using simple material relations and static analysis, an estimate can be obtained for a range of column size and load. However, for complex systems, the prediction of failure can be difficult due to the interaction of many forces and the confounding interaction of many parameters. In these cases, detailed models that govern the fundamental behavior of systems are designed to predict the performance of systems immediately prior to failure. Depending on the application, models can be computationally expensive, and exploration of large design spaces can be cost prohibitive, yet engineers often want to know where in the design space a system will fail.

The calculation of microelectronic device failure due to a single energetic particle strike [1,2] is an example of an application that requires significant computational time for a single design point. Failure of a device is determined by the particle energy, biasing conditions, and strike location and direction, among other parameters. Simulations for a single design point require hours of computer time. As a result, to locate all combinations of bias, particle energy and direction that result in failure would require years of computer time.

Another example that does not deal directly with failure but is a similar class of engineering problem is ignition prediction. In diesel engines, ignition is dependent on a variety of parameters such as temperature, pressure, crank angle, cylinder charge, and so forth. To predict whether ignition will occur, a detailed chemistry and thermodynamic simulation must be performed, allowing the states to evolve in time as the gas is compressed by the cylinder [3]. Failure in this context would actually be ignition, but the results of the simulation indicate whether the fuel mixture burns or not. These models require expensive computations.

These examples fall into a class of problems that are characterized by a binary objective: The diesel fuel ignited or it did not; the ion destroyed the device or it did not; the column buckled or it did not. Furthermore, simulation results on the "unfailed" side are not terribly useful for determining how close to failure we might be; we simply know the system did not fail. Results obtained from design points of the failed side are often unavailable because the simulation entered an error condition or the quantities are nonphysical (due to failure). Therefore, we cannot depend on the simulation results to provide any more information other than failure or not-failure. In essence, we are attempting to find the location of the discontinuity (x_d) in a Heaviside step function

$$H(x) = \int_{-\infty}^{x} \delta(x' - x_d) dx' \tag{1}$$

Contrast the foregoing problem to the discrete optimization problems where the design variable is discrete. In general, discrete optimization problems do not demand a discrete objective. One notable example is the knapsack problem [4]—a simple example of the discrete optimization problem. In this case, the objective is the number of items in the knapsack, which is discrete. Nevertheless, discrete optimization methods, in general, focus on the fact that the design variables are discrete and are less concerned about the continuity of the objective. In our case, the design variables are continuous and the objective is discrete.

Not only is our objective discrete, but it is also binary. However, our problem is not a pseudo-Boolean optimization (also related to binary optimization or zero-one integer programming) either [5]. These techniques are special cases of discrete optimization when the design variable is binary. Again, in general, the objective remains continuous in these classes of problems.

Due to the lack of information from a function evaluation, binary objective problems are amenable only to direct search methods much like those problems without numerical objectives [6]. Problems without numerical objectives still usually have a "better" or "worse" designation between any two points even though the objective is not quantified. This qualitative evaluation still gives a search direction. Our binary objective is similar except that we do not have enough information to determine a direction to proceed. We assume that gradients are nonexistent or meaningless except across the interface where the objective changes by a unit step. Nevertheless, direct search methods can be applied to these problems.

The binary objective problem somewhat resembles that of an edge detection problem in image processing [7]. However, traditional approaches still rely on higher order derivatives to smooth the image and locate where in an image the intensity changes dramatically. The Canny approach [8] assumes that noise is inherent to the data and that all function evaluations have already been acquired (pixel color). These two assumptions are contrary to our problem and make the standard edge detection algorithms impractical for our approach. Moreover, the type of edge being detected is usually quite different from our design space surface. Nevertheless, the interpolation line segment approach used by some edge detection routines [9] is partially responsible for the interpolation scheme used in our approach to speed up serial analysis. Finally, edge detection problems do not scale to multiple dimensions, making these approaches only peripherally related and of limited importance to the present work.

In general, optimization problems are amenable to parallelization. Byrd et al. [10] suggested that either the function evaluations

Contributed by the Simulation and Visualization Committee of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received June 9, 2009; final manuscript received May 7, 2010; published online November 23, 2010. Assoc. Editor: J. Michopoulos.

can be performed concurrently or that a sequential optimization can be used with a parallel forward model. Eldred et al. [11] further developed a taxonomy for levels of parallelization that, when used properly, provides maximum use of available resources. In fact, the DAKOTA project at Sandia National Laboratories was developed to exploit the multilevel parallelism of most optimization problems [12]. In the present context, we assume that the parallelization of the forward model is limited and that more resources exist than can be used by the forward model. Therefore, we are interested solely in the parallelization of the optimization algorithm.

The vast majority of the research on parallelization of optimization algorithms is in the area of smooth nonlinear functions [13]. In these cases, gradients are usually used to provide a new search direction. If analytic gradients are not available, they can be approximated with finite differences. In terms of parallelization, each gradient (or Hessian) requires additional function evaluations that can be performed concurrently. Therefore, the optimal speedup is equivalent to the number of function evaluations required for each iteration [14]. This rule of thumb can be improved slightly with more modern quasi-Newton methods [15,16] and using additional information from otherwise unutilized processors to augment the optimization [14]. Nevertheless, the number of processors that can be used is limited by the dimensionality of the design space. Consequently, for the most practical optimization problems, the amount of parallelization for practical gradient-based optimization algorithms is not massive.

Nongradient-based optimizations, however, stand to see greater gains than their gradient-based counterparts. Chazan and Miranker [17] showed that an order advantage exists between sequential search algorithms and parallelized search algorithms for quadratic functions. Sutti [18] later extended the methods by performing asynchronous evaluations, which simply means that the order of the linearly independent search direction s is immaterial. Nevertheless, these methods depend on a smooth convex function, which is not representative of our binary function.

The binary objective problems described here can be treated as trivially parallel in that the results of each design point are independent of the results from other design points. Therefore, high-performance solutions can greatly reduce the total amount of time an engineer might have to wait for the results. The simplest approach would be to discretize the design space and evaluate each point on a different processor. However, this approach does not limit the search in any way and can also become computationally prohibitive for multidimensional searches or when a limited number of processors is available. In general, the brute force approach scales such as N^p where N is the number of parameters values to be tested for a single parameter and p is the number of parameters to be tested.

The present work focuses on a parallelized and load balanced bisection of the design space to locate a sudden transition at the discontinuity in the objective. We define the mathematical formulation of the problem and solution approach, explore how the solution can be improved serially, parallelize a basic bisection algorithm, and add load balancing. Performance of the serial solution is measured by the number of function evaluations required to obtain the threshold in the objective. Performance of the parallel algorithms is indicated by the speedup.

2 Mathematical Model and Solution Approach

The mathematical model of the problem is described in terms of adjustable parameters that govern the physics given by real variables represented by $z_k \ \forall k \in [1,2,\ldots,n]$. The parameter space is then the set of all possible design points P where $f(z_1,z_2,\ldots,z_n) \rightarrow [0,1]$ is a mapping from P to a binary variable. The problem can be defined as the following: Find sets $V_0 \subseteq P \ni f(z_1,z_2,\ldots,z_n) \rightarrow 0$ and $V_1 \subseteq P \ni f(z_1,z_2,\ldots,z_n) \rightarrow 1$.

To make the problem more tractable, it is assumed that the boundary between the sets can be represented as a continuous $(n-1)$ dimensional surface S in P. Because of this simplifying assumption, the solution may be represented by this surface and a binary value that indicates whether the value 1 lies above or below the surface.

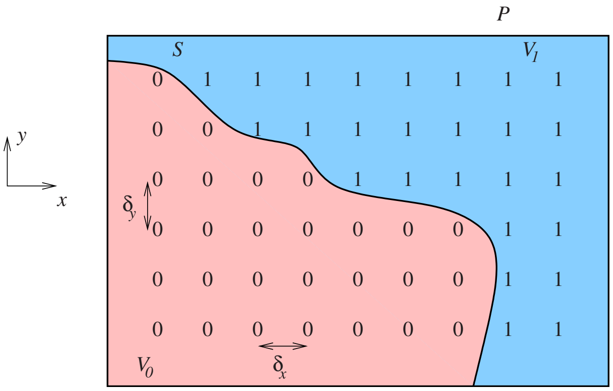

Fig. 1 Illustration of the solution *S* on the domain *P* and the value of the function evaluation *f* for discretized points whose distances (δ_x and δ_y) are determined by a prescribed tolerance

In terms of the computational complexity theory [19], the problem is stated as finding the region of the design space that results in failure (V_0) or success (V_1) to within a particular tolerance such that the surface S is nowhere further from the closest points in either set V_0 or V_1 than a prescribed tolerance. If the continuous design space is discretized using the tolerance, then a naive algorithm would be to evaluate the function f for each discretized design point in the domain P. This algorithm is $O(n^d)$, where n is the number of design points to test in each direction d (assuming the tolerance in each direction is specified to yield the same number of discretized points in each direction). Although we expect to be able to do better, we are not trying to establish a lower bound to the complexity of the problem. Instead, we simply wish to improve on the $O(n^d)$ scaling.

To illustrate the solution process and provide metrics for evaluating the efficacy of the algorithm, a two-dimensional design space ($d=2$) search is considered. Assume that the boundary S between success and failure can be represented by a continuous curve in the $x-y$ domain and that $x \in [x_0,x_1]$ and $y \in [y_0,y_1]$ define the parameter space P. The goal is to find the boundary S within x_{tol} along the x-direction and y_{tol} along the y-direction.

The naive solution (full parameter sweep) proceeds by discretizing the design space using the tolerance for each direction. Each design point can be evaluated to determine whether that point results in failure or success. The boundary S then lies between neighboring design points of opposite results and is never farther from the actual location than the specified tolerance. This algorithm is $O(n_x n_y)$, where n_x and n_y are the numbers of discretized points in each direction that can be computed as

$$n_x = 1 + \text{ceil}\left(\frac{x_1 - x_0}{x_{\text{tol}}}\right) \quad \text{and} \quad n_x = 1 + \text{ceil}\left(\frac{y_1 - y_0}{y_{\text{tol}}}\right) \tag{2}$$

Assuming the tolerances in each direction are commensurate, the algorithm can be said to be $O(n^d)$, where d is the dimensionality or number of design points. An illustration of this strategy is shown in Fig. 1 where the distance between points is

$$\delta_x = \frac{x_1 - x_0}{n_x - 1} \le x_{\text{tol}} \quad \text{and} \quad \delta_y = \frac{y_1 - y_0}{n_y - 1} \le y_{\text{tol}} \tag{3}$$

A better solution proceeds by allowing $x_i = x_0 + i x_{\text{tol}} \ \forall i \ni x_i < x_1$. For each x_i, perform a bisection along the y-direction to bracket the boundary. The number of function evaluations can be

Table 1 Number of function evaluations required to locate the surface *S* identified above to a tolerance of 10^{-4}. The row labels identify the type of interpolating scheme and the column labels identify the type of interface function.

Type	Line	Parabola	Poly-9	Sine
Akima	9336	8153	3870	20,243
Akima-periodic	10,434	10,278	3580	20,859
Cspline	4421	3922	3574	20,859
Cspline-periodic	21,164	5313	3578	20,985
Linear	4421	4694	3574	20,859
None	21,021	21,002	2724	21,021
Polynomial	341,670	320,794	3749	340,084

estimated based on the tolerance required for each direction without having any knowledge of the shape of the surface S. The bisection algorithm gives

$$O\left[\frac{x_1-x_0}{x_{\text{tol}}}\log\left(\frac{y_1-y_0}{y_{\text{tol}}}\right)\right] \tag{4}$$

number of function evaluations. This is a significant improvement over a parametric sweep, which requires

$$O\left(\frac{x_1-x_0}{x_{\text{tol}}}\times\frac{y_1-y_0}{y_{\text{tol}}}\right) \tag{5}$$

function evaluations. Assuming the boundaries and tolerances are the same in each direction, the total number of function evaluations required for finding the surface S is $O(n \log n)$ for the bisection instead of $O(n^2)$ for the full parametric sweep. As the number of dimensions increases, the advantage is not quite as dramatic because the complexity increases as $O(n^{d-1} \log n)$. Nevertheless, a bisection should improve the performance, and alternative algorithms could be used such as a Brent algorithm [20].

In some instances, the bisection could fail to provide an advantage. For example, consider the right most portion of the example provided in Fig. 1. For the two right most columns, the bisection would not bracket the solution because the solution is not there to bracket. Only after the requisite number of evaluations ($O[\log N]$) would we be able to conclude that the solution S does not extend into that region. If we could leverage existing knowledge about the locus of the solution, perhaps we could reduce or eliminate evaluating these points. This is the focus of the approximation scheme described below.

3 Serial Improvements

To reduce the number of function evaluations even further, an approximation scheme can be used, which will leverage information already gathered about the location of the surface S. These approximations serve as a guide for the remaining bisections. Types of approximation used include all of the interpolation types supported by the GNU Scientific Library [21]: linear, polynomial, cubic spline, periodic cubic spline, akima spline, and periodic akima spline interpolations. Each type of interpolation was tested on a boundary represented by a line, sine, parabola, and polynomial. The results in terms of the number of function evaluations are presented in Table 1. The results suggest that simple interpolation (linear) is adequate for the vast majority of function types.

These functions examined are all smooth, and so the results are quite impressive. Cubic splines perform very well, but performance is worse than the regular bisection method in the case of the ninth degree polynomial, which wildly oscillates outside of the parameter space.

4 Parallel Algorithm

This problem can be parallelized easily because each function evaluation is assumed to be independent of all other design points. Therefore, little extra work is required to parallelize the algorithm: One must only assign a single x-value to each node and allow the bisection for that x-value to take place on that node. The number of function evaluations does not change when the algorithm is parallelized in this way.

A parallel version of the search algorithm was developed using MPI (message passing interface) with ethernet network connectivity. No attempt to balance the load was made in this initial trial. Because the number of design points in the x-direction was fixed (based on the tolerance), each processor was assigned a fixed number of evaluations in that direction. For each x design point, bisection was performed in the y-direction until the required tolerance was reached. The test problem was to find the threshold given by $y=0.75x+0.13$ on the domains $x_0=0$, $x_1=1$, $y_0=0$, and $y_1=1$. The model returned a 1 if the design point was above the line and 0 if the design point was below the line. The required tolerances for each direction were $x_{\text{tol}}=10^{-2}$ and $y_{\text{tol}}=10^{-3}$. No interpolation was performed to reduce the number of iterations.

There are three possible sources of task distribution that could prohibit the parallelization from achieving a linear speedup.

1. The number of tasks does not divide easily among the available processors. For example, four function evaluations do not divide equally across three processors. While one processor finishes the final function evaluation, the other three processors sit idle.
2. Each processor may be assigned a different number of function evaluations due to predictive algorithms. For example, if one can provide a good guess where the interface is based on previous results, fewer iterations may be required to achieve the specified tolerance. Therefore, the processors where a predictive algorithm is effective may finish their tasks earlier than other processors.
3. Each function evaluation may require a different amount of time. For example, in the case of the ion-strike simulations described previously, the average time to determine whether a device failed given a single design point is 4 h, yet some failure conditions can be estimated early in the simulation (of the order of 20 min of simulation time), and some simulations that come close to failure but ultimately survive can require 10–20 h of CPU time. Therefore, depending on the number of processors, it is conceivable that several processors may sit idle while the last few simulations finish the calculations.

In the foregoing tests, the number of function evaluations (1200) is much greater than the number of processors (up to 17); therefore, the load based on the number of function evaluations (point 1) should be similar. Concerning point 2, we are not using any predictive algorithm, so the number of function evaluations will not change based on previous results. Finally, if each function evaluation requires the same amount of time, then we expect linear speedup, as shown in Fig. 2, for the constant function evaluation time case. These results were obtained on a homogeneous cluster.

In real engineering systems, however, we cannot depend on each function evaluation to be equivalent in terms of computational requirements (point 3). This wait time at the end of the simulation will reduce the speedup of the overall analysis, as shown in Fig. 2. In the random function evaluation time case, a random amount of sleep time was executed before the function returned with a 1 or a 0. This was implemented to simulate varying function evaluation times. This lack of linear speedup can be addressed through a load balancing scheme.

The strategy used to accomplish load balancing is that of a simple client-server architecture. The server maintains a queue of tasks to be completed. Initially, the queue contains, in order, the low and high y-values for each x-value. The server doles out tasks from the front of the queue. When a task completes, a new task is generated and is placed at the end of the queue. A new task can be

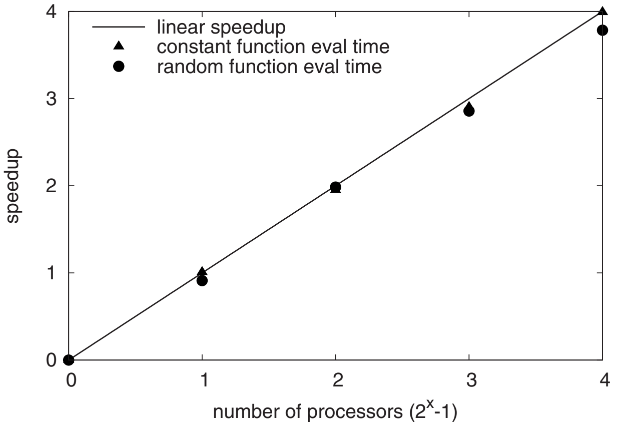

Fig. 2 Speedup measured for a constant function evaluation time and a random function evaluation time (from 0 s to 5 s of sleep)

assigned in constant time. This form of load balancing addresses all three sources of degradation of linear speedup. However, it requires a dedicated server to dole out design points and requires a bit more communication overhead. Nevertheless, the results are dramatically improved. Figure 3 shows the maximum wait time of any processor in the group and the average wait time for all the processors in the group as a function of the number of processors in the group. The amount of wait time increases with the number of processors when no load balancing is performed. With load balancing, however, the wait time is decreased significantly. Therefore, the efficiency of this strategy is nearly perfect, leading to a scalable speedup up to 17 processors.

5 Discussion

The performance of the parallel algorithm can be improved by including the approximation scheme used in the serial algorithm. However, this effort introduces additional complexity. As processors are added to the simulation, the accuracy of the approximation decreases, as illustrated in the following example. If all execution occurs on a single processor, then when the evaluation k is assigned $(k-1)$, other results may be used in the approximation. On the other hand, if ten clients are used, then when the evaluation k is assigned at most $(k-10)$, other results may be used. Furthermore, the approximation in this case is an interpolator, meaning that the approximation must pass through each point. For these reasons, it is extremely difficult to place bounds on the number of evaluations required for a particular number of processors and type of approximation. As illustrated in the case of the ninth degree polynomial, the number of function evaluations does not always decrease when using approximation.

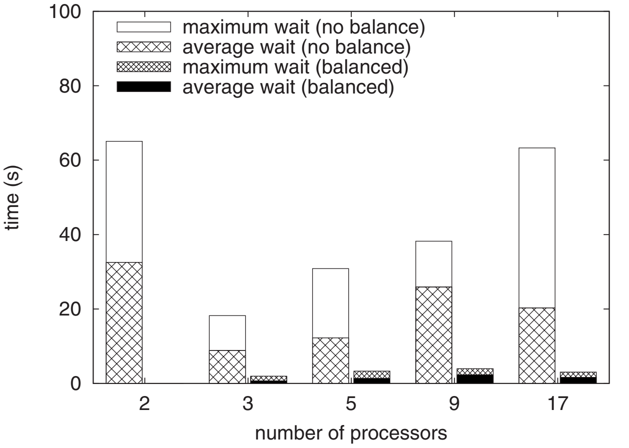

Fig. 3 The maximum and average wait times for a nonbalanced case and when load balancing is employed

Although the example was cast in a two-dimensional space, the concepts here can be extended to multiple dimensions with a multidimensional bisection [22] algorithm. In fact, the two-dimensional problem may also be improved with a multidimensional bisection approach. However, the scalability is no longer a simple function of the problems size, and the parallelization is no longer trivial.

Improvements can be made to the load balancing strategy by introducing a list of

$$\log\left(\frac{y_1 - y_0}{y_{\text{tol}}}\right) \tag{6}$$

queues q_i such that q_i contains only tasks corresponding to an x-value that has i tasks executed already. This strategy is commonly used in CPU schedulers and runs in constant time.

Other improvements could include mixing an approximation strategy with a load balancing strategy. This is difficult because the approximation and thus the scalability worsen as the number of processors increases. Also, the interpolators used should be replaced with better approximations since the partial results are inexact at best.

6 Conclusions

Binary objectives occur naturally in many areas of science and engineering. Therefore, understanding these types of problems is important to the efficient analysis and designs of engineering systems. Examining the problems by forming a mesh of the parameter space seems to be the accepted solution, which is certainly easy to extend to n dimensions. However, complicated physical models often require vast computing resources, and an algorithm that scales better and reduces the computational cost is essential. We have shown that a simple bisection algorithm and simple parallelization with load balancing can reduce the computational cost by an order of magnitude. These results are limited to a boundary that is continuous in the parameter space. Moreover, we did not test the scheme for higher dimensions. Nevertheless, these results lay the framework for a continued study in optimization problems with binary objectives.

References

[1] Fleetwood, D. M., Winokur, P. S., and Dodd, P. E., 2000, "An Overview of Radiation Effects on Electronics in the Space Telecommunications Environment," Microelectron. Reliab., **40**(1), pp. 17–26.

[2] Schrimpf, R. D., Weller, R. A., Mendenhall, M. H., Reed, R. A., and Massengill, L. W., 2007, "Physical Mechanisms of Single-Event Effects in Advanced Microelectronics," Nucl. Instrum. Methods Phys. Res. B, **261**, pp. 1133–1136.

[3] Kong, S. C., and Reitz, R. D., 1993, "Multidimensional Modeling of Diesel Ignition and Combustion Using a Multistep Kinetics Model," ASME J. Eng. Gas Turbines Power, **115**, pp. 781–789.

[4] Gallo, G., Hammer, P. L., and Simeone, B., 1980, "Quadratic Knapsack Problems," Math. Program., **12**, pp. 132–149.

[5] Boros, E., and Hammer, P. L., 2002, "Pseudo-Boolean Optimization," Discrete Appl. Math., **123**(1–3), pp. 155–225.

[6] Kolda, T. G., Lewis, R. M., and Torczon, V., 2003, "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods," SIAM Rev., **45**(3), pp. 385–482.

[7] Marr, D., and Hildreth, E., 1980, "Theory of Edge Detection," Proc. R. Soc. London, Ser. B, **207**(1167), pp. 187–217.

[8] Canny, J., 1986, "A Computational Approach to Edge Detection," IEEE Trans. Pattern Anal. Mach. Intell., **PAMI-8**(6), pp. 679–698.

[9] Kass, M., Witkin, A., and Terzopoulos, D., 1988, "Snakes: Active Contour Models," Int. J. Comput. Vis., **1**(4), pp. 321–331.

[10] Byrd, R. H., Schnabel, R. B., and Schultz, G. A., 1988, "Using Parallel Function Evaluations to Improve Hessian Approximation for Unconstrained Optimization," *Parallel Optimization on Novel Computer Architectures*, Annals of Operations Research, P. L. Hammer, R. R. Meyer, and S. A. Zenios, eds., J. C. Baltzer AG Scientific, Basel, Switzerland, Vol. 14, pp. 167–193.

[11] Eldred, M. S., Hart, W. E., Schimel, B. D., and van Bloemen Waanders, B. G., 2000, "Multilevel Parallelism for Optimization on MP Computers: Theory and

Experiment," Proceedings of American Institute Aeronautics and Astronautics, Paper No. AIAA-2000-4818.

[12] Eldred, M. S., Hart, W. E., Bohnhoff, W. J., Romero, V. J., Hutchinson, S. A., and Salinger, A. G., 1996, "Utilizing Object-Oriented Design to Build Advanced Optimization Strategies With Generic Implementation," Sixth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Washington, DC.

[13] Gill, P. E., Murray, W., and Wright, M. H., 1981, *Practical Optimization*, Academic, New York.

[14] Schnabel, R. B., 1987, "Concurrent Function Evaluations in Local and Global Optimizations," Comput. Methods Appl. Mech. Eng., **64**, pp. 537–552.

[15] Byrd, R. H., Schnabel, R. B., and Shultz, G. A., 1988, "Parallel Quasi-Newton Methods for Unconstrained Optimization," Math. Program., **42**, pp. 273–306.

[16] van Laarhoven, P. J. M., 1985, "Parallel Variable Metric Algorithms for Unconstrained Optimizations," Math. Program., **33**, pp. 68–81.

[17] Chazan, D., and Miranker, W. L., 1970, "A Nongradient and Parallel Algorithm for Unconstrained Minimization," SIAM J. Control, **8**(2), pp. 207–217.

[18] Sutti, C., 1983, "Nongradient Minimization Methods for Parallel Processing Computers," J. Optim. Theory Appl., **39**(4), pp. 475–488.

[19] Papadimitriou, C. H., 2003, "Computational Complexity," *Encyclopedia of Computer Science*, 4th ed., Wiley, New York, pp. 260–265.

[20] Brent, R. P., 1973, *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ.

[21] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., and Rossi, F., 2001, *GNU Scientific Library Reference Manual*, 3rd ed., Network Theory Limited, Bristol, UK.

[22] Wood, G. R., 1992, "The Bisection Method in Higher Dimensions," Math. Program., **55**, pp. 319–337.