Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–21

RESEARCH ARTICLE

Asynchronous parallel hybrid optimization combining DIRECT and GSS

Joshua D. Griffin[a] and Tamara G. Kolda[b*]

[a]*SAS Institute Inc., 100 SAS Campus Drive, Cary, NC 27513, USA*; [b]*Informatics and Decision Sciences Department, Sandia National Laboratories, Livermore, CA 94551, USA*

(*Received 00 Month 200x; in final form 00 Month 200x*)

In this paper we explore hybrid parallel global optimization using DIRECT and asynchronous generating set search (GSS). Both DIRECT and GSS are derivative-free and so require only objective function values; this makes these methods applicable to a wide variety of science and engineering problems. DIRECT is a global search method that strategically divides the search space into ever-smaller rectangles, sampling the objective function at the center point for each rectangle. GSS is a local search method that samples the objective function at trial points around the current best point, i.e., the point with the lowest function value. Latin hypercube sampling (LHS) can be used to seed GSS with a good starting point. Using a set of global optimization test problems, we compare the parallel performance of DIRECT and GSS with hybrids that combine the two methods. Our experiments suggest that the hybrid methods are much faster than DIRECT and scale better when more processors are added. This improvement in performance is achieved without any sacrifice in the quality of the solution — the hybrid methods find the global optimum whenever DIRECT does.

Keywords: parallel, asynchronous, distributed computing, hybrid optimization, global optimization, direct search, derivative-free, generating set search (GSS), pattern search

AMS Subject Classification: 90C56, 90C30, 65K05, 65Y05, 68W10

1. Introduction

We consider the problem of global optimization for computationally expensive black-box objective functions with a small number of parameters (say, less than 10). Such problems often arise in science and engineering when the objective function is based on a complex computer simulation used to model, for example, a physical phenomenon that is difficult or expensive to replicate in an experiment. These simulations can be expensive, requiring minutes, hours, or even days to complete individual runs. The code for such simulations is oftentimes so complex that we treat it as a "black box" for the purposes of optimization; as a result, we do not have access to detailed information about the objective function (e.g., smoothness, continuity, computational error, gradient information). Moreover, the function evaluation may fail unexpectedly, in which case we assign the objective

This work was funded by Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.
*Corresponding author. Email: tgkolda@sandia.gov

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/03081080xxxxxxxxx
http://www.informaworld.com

value to be $+\infty$. Mathematically, the problem we considered is of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{subject to} \quad & \ell \leq x \leq u. \end{aligned} \tag{1}$$

Here $f : \mathbb{R}^n \to \{\mathbb{R}, +\infty\}$ is the objective function (incorporating any linear or nonlinear constraints via suitable penalty functions), and finite upper and lower bounds exist for each variable with $\ell_i < u_i$ for $i = 1, \ldots, n$. Because the characteristics of the objective function are unknown, the global optimization method we propose is *incomplete* [49] in that it uses clever heuristics but cannot ensure that it will find a global optimum; however, if information about the continuity of the function is provided, asymptotic guarantees of a global optimum may be possible. Our specific focus is on the use of distributed cluster of computers, where multiple function evaluations may be computed in parallel. The limiting factor for our intended users is wall-clock time, e.g., the user wants the best answer that can be computed in a pre-designated amount of time.

The nature of the target problems and our desire for parallel function evaluations influences our choices of optimization methods. First, the presence of noise means that we prefer direct search methods because they do not attempt to directly use, estimate, or model derivatives [38]. Second, since the function evaluations are expensive, metaheuristics such as genetic algorithms and simulated annealing [23, 37, 41] may be too expensive because they require a large number of function evaluations and are appropriately mainly for very inexpensive function [49]. Third, surrogate-based optimization methods (e.g., [52, 55]) are not easily parallelized, though surrogates can and have been incorporated into the computational framework we propose, as discussed in more detail below.

Having ruled out derivative-based methods, metaheuristics, and surrogates, we focus on sampling methods. DIRECT [36] is a well-known global optimization method that is already popular for this class of problems. DIRECT strategically refines the search space into ever-smaller rectangles, balancing local and global refinement. The drawback of DIRECT is that it has poor load balancing and sometimes has problems identifying the exact solution even if it is close to it. Another popular method is generating set search (GSS) [38], and we specifically consider its asynchronous parallel implementation in the APPSPACK software package [23, 37]. This method is attractive because it has excellent parallel load balancing. The drawback of GSS is that, even though it is often used to solve global optimization problems, it was not designed for that purpose. Our goal here is to combine the strengths of these two methods to create an asynchronous, parallel, hybrid method for reliably finding good solutions to global optimization problems.

Our framework generalizes the parallel hybrid model implemented in Gray et al. [26] that used Treed Gaussian Process Models (TGP) and GSS simultaneously for electronic calibration problems. In their approach, TGP surrogate models are constructed from the cache of evaluated points. These models are used to predict new trial points that either reduce the objective function or reduce uncertainty in the corresponding model. The trial points produced by TGP are also fed to GSS, which conducts a local search around the current best point. Both GSS and TGP trial points are evaluated asynchronously in parallel using an adaptation of the APPSPACK software [23]. See also [5] for the earliest example of combining generalized pattern search (a variant of GSS) with a surrogate in a general framework. We note also that our approach is similar to [60, 61] in which DIRECT is followed by several iterations of multidimensional search or generalized pattern search (both

variants of GSS). In our case, though, the two methods are run simultaneously.

The paper is organized as follows. In §2, we review DIRECT (§2.1), asynchronous parallel GSS (§2.2), and LHS (§2.3). In §3, the new parallel hybrid approach that combines DIRECT and GSS is described. In §4, we review related work and describe our approach in the context of hybrid optimization, using the categorization method proposed by Raidl [54]. Numerical results on a set of global optimization test problems are provided in §5, and the conclusions and future work are discussed in §6. Details of the test problems are provided in Appendix A.

2. Background

Without loss of generality, in the discussion that follows, we assume that (1) has been transformed to the unit hypercube so that $\ell = \mathbf{0}$ (the vector of all zeros) and $u = \mathbf{1}$ (the vector of all ones). This can be accomplished by affinely transforming the variables to get $\tilde{x} = S^{-1}(x - \ell)$ with $S = \text{diag}(u_1 - \ell_1, \ldots, u_n - \ell_n)$. Then (1) becomes

$$\begin{aligned} \min_{\tilde{x} \in \mathbb{R}^n} \quad & \tilde{f}(\tilde{x}) = f(S\tilde{x} + \ell) \\ \text{subject to} \quad & \mathbf{0} \leq \tilde{x} \leq \mathbf{1}. \end{aligned} \tag{2}$$

2.1. *DIRECT*

Jones, Perttunen, and Stuckman [36] developed DIRECT, short for Dividing Rectangles, as a derivative-free, deterministic sampling algorithm for global optimization on a bound-constrained region. Subsequently, others have analyzed, enhanced, and tested the method; see, e.g., [3, 9, 10, 15–17, 20, 21, 35, 43, 57]. DIRECT systematically divides the feasible region into hyperrectangles of decreasing size and evaluates the objective function at the hyperrectangle centers. DIRECT strategically balances local search (subdividing hyperrectangles whose centers have the smallest function values) and global search (subdividing the largest hyperrectangles) to find the global minimum efficiently. If the objective function is continuous in a neighborhood of a global optimum, DIRECT converges to the globally optimal function value [36]. DIRECT is a complete [49] method in the sense that it reaches a global minimum with certainty assuming exact computations and an indefinitely long run time and knows (assuming knowledge of the size of the neighborhood on which f is continuous and the Lipschitz constant on that region) after a finite time that an approximate global minimizer has been found. In our case, since nothing is known about the properties of the objective function, DIRECT is incomplete [49], i.e., just a clever heuristic. There has been some analysis of DIRECT in the nonsmooth case [16], but there is no guarantee of global optimality.

We refer the reader to the references above for the details of DIRECT, and Algorithm 1 gives a high-level summary of the algorithm. We assume that the bound-constrained region has be scaled as in (2) so that all the edges of the initial hyperrectangle are the same length. At each iteration, a subset of the hyperrectangles are selected for subdivision. Hyperrectangles are always divided along their long edges first, so that a single hyperrectangle has at most two different edge lengths. The function values at what will be the new hyperrectangle centers are computed, and the selected hyperrectangles are divided. Figure 1 shows what three iterations of DIRECT might look like for a two-dimensional problem. In the first iteration, there is only one hyperrectangle and it is split. There is some choice as to how to create the splits (the long boxes could be horizontal rather than vertical),

Algorithm 1 DIRECT

Let $\mathcal{H}$ denote the set of hyperrectangles (initially the entire domain). Evaluate the center point of the domain (the sole initial hyperrectangle). While the function evaluation budget is not exhausted, repeat the following:

(1) Choose a set $\mathcal{S} \subseteq \mathcal{H}$ of "potentially optimal" hyperrectangles to subdivide.
(2) For each hyperrectangle in $\mathcal{S}$, do the following.
   a) Let c denotes its center. Let $\mathcal{I} \subseteq \{1, \ldots, n\}$ denote the dimensions of the long edges of the n-dimensional hyperrectangle.
   b) Compute $f(c + \delta e_i)$ and $f(c - \delta e_i)$ for each $i \in \mathcal{I}$ where δ is 1/3 the length of the long edge.
   c) Based on the function values just computed, divide the hyperrectangle. The points $c \pm \delta e_i$ for $i \in \mathcal{I}$ are the centers of the new hyperrectangles.

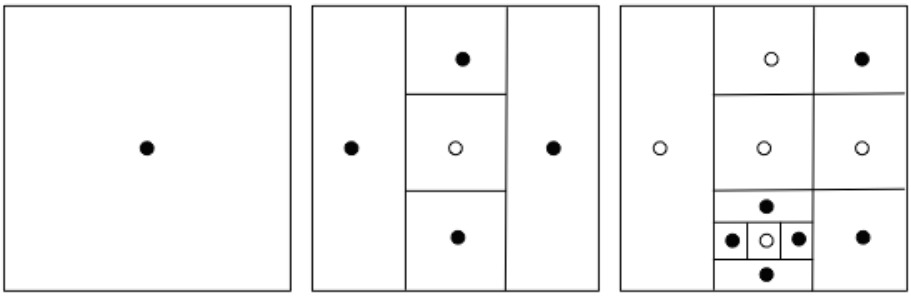


Figure 1. DIRECT strategically subdivides one or more hyperrectangles at each iteration. Here black circles represent new points to be evaluated and while circles represent previously evaluated points.

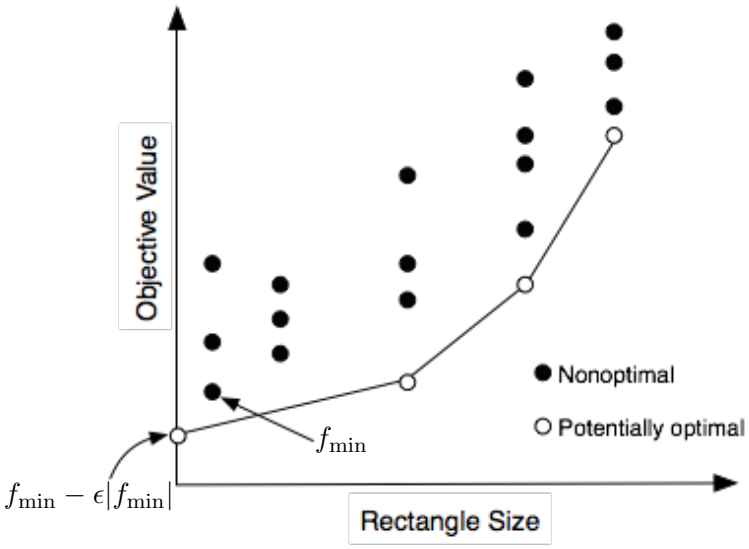


Figure 2. Potentially optimal hyperrectangles are selected by computing the lower convex hull on a plot of rectangle size versus objective value. In this case, only three rectangles are selected to be subdivided (the open circle on the y-axis corresponds to an imaginary ideal hyperrectangle).

which is discussed in detail in [36]. The result is shown in the center figure. At this point, two hyperrectangles are selected for subdivision. For each hyperrectangle that is subdivided, up to $2n$ function values are required. Choosing which hyperrectangles to subdivide at a given iteration (in Step 1) requires computing the lower convex hull on a plot of rectangle size versus the function value at the center. An example is shown in Figure 2. Here we see the points $(d, f(c))$ for every

hyperrectangle where d is a measure of the hyperrectangle size (specifically the distance from the center to a vertex, which is the same for all vertices), and $f(c)$ is the function value at the center. The method also plots the point $(0, f_{\min} - \epsilon|f_{\min}|)$ where $f_{\min}$ is the minimum function value discovered so far and ϵ is a user-defined parameter. The "potentially optimal" hyperrectangles are those that fall on the lower convex hull on the these points, as illustrated in the figure by white circles. Smaller values of ϵ cause the search to be more local while larger values make it more global. In our experiments, we use $\epsilon = 0.01$ because it is the value originally used by Jones et al. [36].

DIRECT is a robust method because it works with black-box function evaluations and is insensitive to discontinuities, missing values, and other common problems. It is also easily parallelized because it computes multiple function evaluations per iteration. Therein, however, also lies one of its drawbacks. Because the number of hyperrectangles divided at each iteration can change, the number of function values at each iteration is also changing, and thus parallel load balancing is a challenge. Moreover, DIRECT can sometimes require a large number of function evaluations to reach a global minimum, especially if the minimizer is on a hyperrectangle boundary. Hybridizing DIRECT will help us to overcome some of these weaknesses.

2.2. *Asynchronous parallel GSS*

GSS [38] is a class of derivative-free optimization methods that search the parameter space according to a specified search pattern (usually the plus and minus unit directions) and a changing step size. In two dimensions, the simplest example is compass search. At each iteration, the search samples north, south, east, and west from the current iterate according to the distance specified by the step length. If an improved point is found, it becomes the next iterate. Otherwise, the step length is halved. Typically the procedure is halted when the step length becomes sufficiently small. Algorithm 2 shows a basic GSS algorithm; we refer the reader to [38] for details. Note that $2n$ function evaluations are required at each iteration, which makes the method easy to parallelize. GSS has been extended to handle linear [40, 44] and nonlinear constraints [39, 45]. GSS is a local optimization method, meaning that it is guaranteed to globally converge to a KKT point if the objective function is sufficiently smooth (i.e., continuously differentiable on the feasible region defined by box constraints).

Algorithm 2 A basic GSS algorithm for a bound constrained problem

We are given x_0 (the initial starting point), Δ_0 (the initial step size), and $\alpha > 0$ (the sufficient decrease parameter). We let $\mathcal{D}$ be the plus and minus unit directions. Each iteration proceeds as follows.

(1) Generate trial points $Q_k = \{x_k + \tilde{\Delta}_k d_i \Big| \ 1 \le i \le |\mathcal{D}|\}$ where $\tilde{\Delta}_k \in [0, \Delta_k]$ denotes maximum feasible step along d_i.
(2) Evaluate all trial points in Q_k
(3) If there exists a point $y \in Q_k$ such that $f(y) < f(x_k) - \alpha\Delta_k^2$, then $x_{k+1} = y$. *Successful iteration.*
(4) Otherwise $x_{k+1} = x_k$ and $\Delta_{k+1} = \Delta_k/2$. *Unsuccessful iteration.*
(5) $k = k + 1$.

Asynchronous parallel GSS is a variant of GSS that does not wait for all trial points to be evaluated (in Step 2) before proceeding to the next iteration [37]. This method has been implemented in APPSPACK [23] and extended to handle

linear [28] and nonlinear constraints [27]. This method can also be shown to globally converge to KKT points if the underlying objective function is continuously differentiable on the box-constrained feasible region. It has the advantage of better load balancing and can significantly reduce parallel run-times [27, 37]. Therefore, we use the asynchronous version in the experiments in this paper.

GSS (and particularly its asynchronous implementation in APPSPACK) is a popular class of methods because it does not require derivative information, nor does it explicitly attempt to model the underlying function. GSS thus tends to be more robust for difficult optimization problems (that may be noisy, with occasional non-smooth, discontinuous, and or undefined "feasible" points), than derivative or model based approaches that break down at points where the function or derivative cannot be computed. APPSPACK has been used to solve a wide variety of problems; see, e.g., [7, 8, 12, 30, 31, 42, 46, 48, 50, 65]. Although GSS is a local solver, it is often used for solving global optimization problems. Our results demonstrate that GSS can be extremely effective on global optimization problems. Past work by Gray et al. [24] showed that GSS is competitive with simulated annealing on a transmembrane protein structure prediction problem. Recently, both GSS (implemented in APPSPACK) and DIRECT were compared directly on a set of groundwater modeling problems and APPSPACK converged more quickly to the best known solution [18]. Therefore, the motivation for hybridizing GSS is that it is already used for global optimization. Hybridizing GSS with a method such as DIRECT will make it more robust for global optimization.

Before we conclude this section, we note that there is one feature of GSS that makes it particularly amenable to hybridization — it can "jump" to a point with a lower function value at any iteration. The easiest way to consider this is that it restarts its search. Algorithmically, we assume we get back Q_k as well as extra external points E_k in Step 2. Then we modify Step 3 to say:

> If there exists a point $y \in Q_k$ such that $f(y) < f(x_k) - \alpha\Delta_k^2$ *or a point* $y \in E_k$ *such that* $f(y) < f(x_k)$, then $x_{k+1} = y$.

It is also a good idea to reset the step length Δ_k to some nominal value whenever an external point is selected as the next iterate because this prevents the step lengths from becoming too short too quickly. This equates to restarting the method whenever an improved external point is discovered. This can be thought of as a "search" step in the generalized pattern search framework discussed in [5].

2.3. *Latin hypercube sampling*

Latin hypercube sampling (LHS) is a form of constrained Monte Carlo sampling developed by McKay, Conover, and Beckman [47] to improve the accuracy of statistical measurements (such as expected values and variable correlations) in higher dimensions using a minimal number of function evaluations. LHS has since become a popular mechanism to aid in the analysis and construction of computer simulation models. In-depth descriptions of LHS can be found in [29, 34, 62].

From an optimization standpoint, LHS is attractive because it is a straightforward algorithm for generating trial points with desirable statistical properties. Further, the user may choose the number of desired trial points independent of problem dimension. To put this in perspective: a naive approach to exploring the feasible region is to use coarse-grid sampling where each dimension is divided into, say, ten equal regions and the grid points are sampled. In two dimensions such a partition results in 121 trial points, but in ten dimensions this results in 11^{10} sample points. LHS, in contrast, is a more efficient mechanism to effectively explore

the feasible region. Figure 3 shows the points that LHS might select as compared to the points that a grid-based sampling would choose.

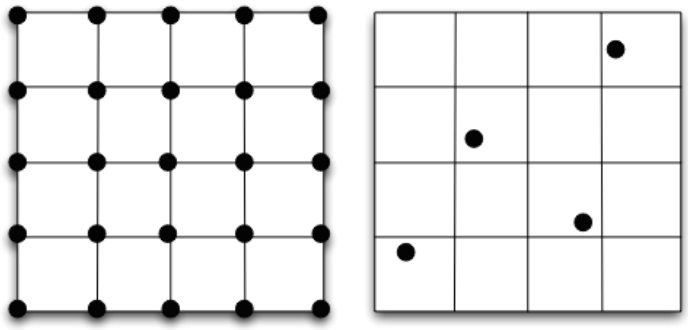


Figure 3. Grid and LHS sampling.

From an optimization point of view, LHS is a tool to get an initial general state. The results of LHS might be used to estimate characteristics of a given objective function such as smoothness. In our situation, we use LHS as a mechanism for sampling more points that DIRECT and GSS would do one their own and to, therefore, assist GSS in its search. There are, of course, other sampling strategies which may be used instead of LHS; see [14, 56] for comparisons of different methods.

3. Hybrid optimization using parallel solvers (HOPS) algorithm

Our goal in using hybridization is twofold. First, better parallel load balancing will be achieved by combining the trial points from all solvers into a single evaluation queue so that they can be efficiently distributed to worker processes. Second, each solver will see the results of all trial point evaluations and can potentially use that information to improve its own search.

The hybrid solver is comprised of a set of P individual *citizen* solvers. Each individual citizen maintains its own evaluation queue, denoted $\mathcal{Q}_p$. At each iteration, HOPS assembles the individual queues into a global evaluation queue, denoted $\mathcal{Q}$. A subset of points are removed from $\mathcal{Q}$ (and the corresponding per citizen queues) for evaluation. The number of points selected depends on how many worker processors are available. The set of evaluated points are returned in the set denoted by $\mathcal{R}$. The hybrid solver is asynchronous because it does not wait for all the points in $\mathcal{Q}$ to be evaluated, but rather returns points to the citizen solvers as the evaluations complete. A description of the algorithm is shown in Algorithm 3 and an illustration of the process is shown in Figure 4.

3.1. *Citizen solvers*

Each citizen solver is an independent entity which may be complex or simple. Here we discuss the four citizens used in the experiments in this paper and shown in Figure 4.

The **LHS** citizen generates a single queue of trial points at initialization using the methodology described in §2.3. Its queue is slowly exhausted as the iterations of the hybrid solver progress. It entirely ignores the returned points ($\mathcal{R}$), never generates any new points, and never deletes any points from its queue.

Algorithm 3 Hybrid optimization using parallel solvers (HOPS)

We are given P solver "citizens" and initialization information for each solver. Each citizen has an associated evaluation queue, $\mathcal{Q}_p$, that is initially empty. At each iteration, we also create a return set of evaluated points, $\mathcal{R}$, that is initially empty. Each iteration of the hybrid method proceeds as follows:

(1) For each citizen ($p = 1, \ldots, P$)
   a) Send the set of *all* evaluated points, $\mathcal{R}$, to citizen p.
   b) Allow the citizen to push points onto the end of $\mathcal{Q}_p$ and to pop off points (from anywhere in $\mathcal{Q}_p$) that are no longer needed.

(2) The *mediator* assembles all the points into a global queue, $\mathcal{Q}$, by interleaving the points in $\mathcal{Q}_1$ through $\mathcal{Q}_P$.

(3) For every available worker, the *conveyor* pops a point from the front of $\mathcal{Q}$ and submits it for evaluation.

(4) The mediator disassembles any remaining points in the global queue, $\mathcal{Q}$, into the individual queues, $\mathcal{Q}_1$ through $\mathcal{Q}_P$, to be updated by the citizens in the next iteration.

(5) The mediator retrieves the set of evaluated points for this iteration in $\mathcal{R}$.

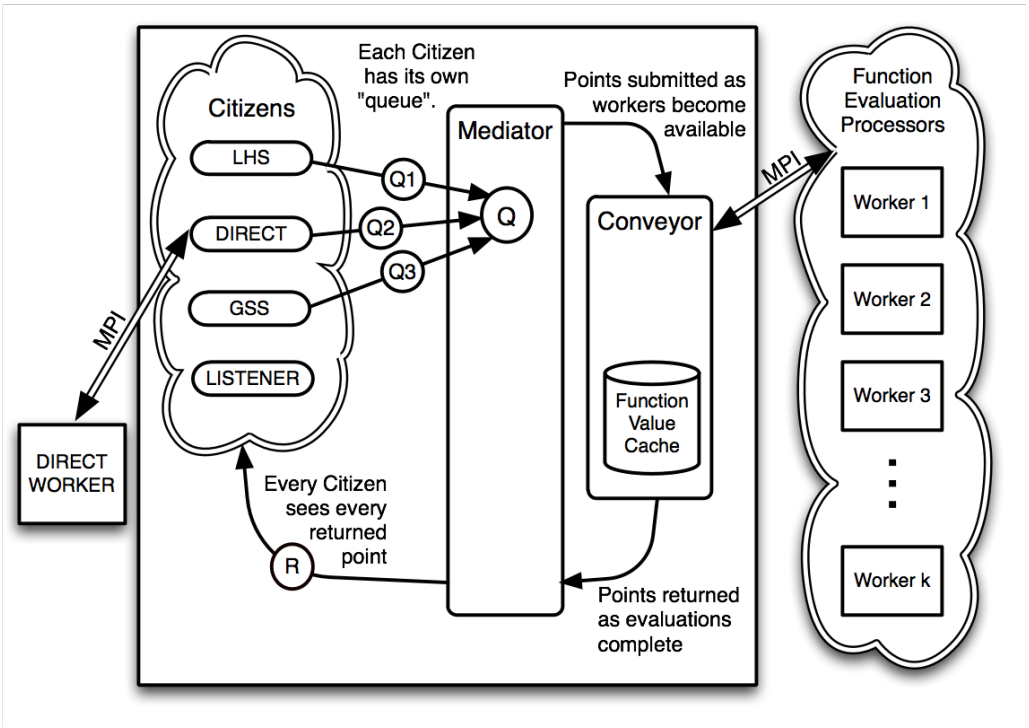


Figure 4. HOPS Illustration. Points are submitted and evaluated asynchronously; no method can force another to wait. Evaluation processors never run idle unless all citizens have submitted empty trial point queues.

The **DIRECT** citizen corresponds to a synchronous iterative global optimization method as described in §2.1. Moreover, our implementation runs DIRECT as a separate (parallel) process, enabling us to use an implementation in a different language. We use an open-source Fortran implementation of DIRECT by Gablonsky [19–21]. This version of DIRECT takes a pointer to an "evaluation function", which in our case passes the set of points back to the main process and the mediator, as described in Algorithm 3 and illustrated in Figure 4. The only minor

modification made to Gablonsky's DIRECT code is the addition of a mechanism to cause it to exit when given an external signal. We note that it is possible to implement DIRECT directly as a citizen within HOPS, but here we demonstrate how existing code can be incorporated using MPI communication. We also observe that we are using a synchronous algorithm within an asynchronous point evaluation framework. Further, we note that the DIRECT citizen ignores all the evaluations that were done for other citizens — it only uses the function values corresponding to its own trial points.

The **GSS** citizen corresponds to asynchronous GSS, as described in §2.2. At each iteration, it adds and deletes points to its queue, depending on the new evaluations it receives. It is the only citizen that can use the information generated by the other citizens. As discussed previously, it can greedily shift its search to whatever point is best so far, regardless of how that point was discovered.

The **Listener** citizen does not generate any points to be evaluated, so its queue is always empty. This citizen is used as a reporting agent in our experiments because it sees all the points that have been evaluated. We mention it as an extreme example of what a "citizen" can be.

The different citizens are summarized in Table 1. At least one citizen must use the points generated by the other citizens for the hybrid method to be effective. In this case, the GSS citizen serves that role.

Table 1. Features of different types of citizens.

Feature	**LHS**	**Direct**	**GSS**	**Listener**
Generates points to be evaluated	Yes	Yes	Yes	No
Uses its own points	No	Yes	Yes	N/A
Uses points from other citizens	No	No	Yes	N/A

3.2. *Evaluating trial points*

Trial points are evaluated using the conveyor mechanism from APPSPACK, as described in detail in [23]. Here we highlight its features and describe a new modification. The general purpose of the conveyor is to pop points from the evaluation queue, assign them to worker processes, and collect and return the results as they become available. Each function evaluation is computationally expensive (requiring minutes, hours, or days to complete) and the time to complete the evaluation varies according to the inputs, computational loads on the processors, random starting conditions, etc.

A particular feature of the conveyor is that it does not reevaluate the same point twice — all function evaluations are stored in a cache for future reuse. (In fact, the cache can even be saved and reused across multiple runs.) For cache look-ups, we consider points x and y to be the same if

$$\|x_i - y_i\| \leq \gamma s_i, \quad \text{for } i = 1, \ldots, n.$$

Here s is a scaling vector that is set to $s = u - \ell$, and γ is a user-defined tolerance that is set to $\gamma = 10^{-5}$ in our experiments.

The procedure that the conveyor follows is shown in Algorithm 4. The function value cache is stored as a splay tree, which is described in detail in [23, 32]. Our additional modification is to also avoid re-evaluating points that are currently being evaluated.

As evaluations complete, the worker processes are marked as free, the function values are cached, and additional identical points also have function values assigned, and the values are returned in $\mathcal{R}$.

Algorithm 4 Conveyor submits points for evaluation

While one or more free worker processes is available:

(1) Pop x off the front of $\mathcal{Q}$.
(2) Check the **function evaluation cache** to see if the point has already been evaluated. If so, retrieve $f(x)$ from the cache and add the evaluated point to $\mathcal{R}$.
(3) Check the **pending evaluation cache** to see if the point has already been assigned to a process. If so, wait until that evaluation completes and return the value for this point as well.
(4) Otherwise, assign the point x to a free worker process for evaluation and mark the worker process as busy.

3.3. *Stopping conditions*

The problem of when to stop a hybrid solver is complicated because each citizen may have its own stopping criterion. We have the option of waiting for every citizen to request a stop, of waiting for a particular solver, or of imposing some external condition. In our case, we use an external condition. We let the hybrid method iterate until either the function evaluation budget is exhausted or the objective function reaches a pre-defined target.

Each citizen may also stop at some point. In our case, the GSS method will stop when its step length reaches a given tolerance (i.e., it has converged to a local minimum). It will only continue again if an improved point is found by another citizen, which essentially restarts the method. The details of software structure and implementation of HOPS will be described in a forthcoming publication.

4. HOPS in the context of hybrid optimization

Hybrid optimization, a popular approach in the combinatorial optimization community, combines metaheuristics such as genetic algorithms, tabu search, ant colony, and variable neighborhood search to improve robustness of the methods, seeking to combine the distinct strengths of different approaches [1]. Recently, metaheuristics have been combined with deterministic methods such as pattern search to form hybrids that perform global and local search; see, e.g., [2, 6, 13, 26, 51, 59, 60, 64, 66]. As mentioned in §1, the new approach described in this paper generalizes Gray et al. [26]; see also closely related work in [25, 58]. Following Puchinger and Raidl [53] and Raidl [54], we describe how HOPS fits into the general class of hybrid optimization according to the following characteristics: 1) the level of hybridization (i.e., the degree of coupling between the methods), 2) the order of execution (interleaved or sequential), and 3) the control strategy (integrative or collaborative).

4.1. *Level of hybridization*

Hybrid methods can be loosely or tightly coupled. Loosely coupled means that the algorithms retain their own identities, whereas tightly coupled algorithms exchange specific low-level information [53]. HOPS is loosely coupled, allowing each solver citizen to retain its own identity and run independently of the others. In fact, as illustrated in the case of DIRECT, it is possible to combine existing software into the HOPS framework with little or no modification. The individual solver citizens are interchangeable. This is beneficial from both a software development and theoreti-

cal perspective because it avoids the problem of reimplementing existing methods and enables us to retain the convergence theory of the individual methods.

4.2. *Order of execution*

The solvers in a hybrid method can be executed one by one (also known as batch or sequential execution) or have their executions interleaved [53]. Using the results of one solver as the starting point for another is an example of a sequential hybrid. In [60], a few iterations of DIRECT are followed by a few iterations of generalized pattern search (a special case of GSS). Other examples of integrating GSS into a sequential hybrid algorithm may be found in [51, 59, 66]. In our strategy, the executions are interleaved, though it is not the case that the iterations of the individual solvers are necessarily interleaved. For example, DIRECT has to wait until all the points from its previous iteration are complete before it can perform its next iteration and generate new points. Asynchronous GSS, on the other hand, performs an iteration for every iteration of the hybrid.

4.3. *Control strategy*

Finally, hybrid algorithms may be viewed as integrative or collaborative [53]. In an integrative approach, one method is subordinate to another; for example, an inner optimization algorithm refines candidate solutions discovered in an outer loop. In collaborative optimization, both algorithms are given equal importance and control. Collaborative approaches may, for example, combine homogeneous algorithms that operate in separate regions of the domain. Our approach is collaborative because each citizen runs independently of the others. As mentioned earlier, GSS has a built-in mechanism to accept trial points from other sources (without effecting its local convergence theory); hence the steady stream of external DIRECT and LHS are easily incorporated into the GSS method.

5. Numerical results

In order to evaluate the effectiveness of HOPS, we compare hybrid and non-hybrid methods on a set of global optimization test problems. Our computational platform is the "Shasta" high-performance computing environment at Sandia National Laboratories. Each compute node has a Dual 3.06 GHz Intel Xeon processor and 2 GB of RAM. We run two MPI processes per node because of the dual processors. We test with 8, 16, and 32 processors (corresponding to 4, 8, and 16 nodes, respectively).

Our test problems comprise nine functions from the original DIRECT experiments [36]. To add some measurable expense to the cost of evaluating the function, we randomly waited between one and five seconds before completing each evaluation; this is a strategy that has been used in our past research as well [28, 33]. Details of the test problems, including the global minimums, are provided in Appendix A. We use the standard stopping criteria based on *percent error* [4, 36]. Specifically, if f_* is the global minimum, then the current iterate x must satisfy

$$100\,\frac{f(x) - f_*}{|f_*|} < 0.01. \tag{3}$$

A trial is considered successful if it satisfies (3) when HOPS terminates.

We compared the following four methods, all implemented within the HOPS framework:

- GSS (G);
- DIRECT (D);
- GSS and DIRECT (GD); and
- LHS, GSS, and DIRECT (LGD).

GSS and DIRECT are standard methods and their implementation within DIRECT does not change their performance. The other two methods (GD and LGD) are hybrids. In GSS (both for single and hybrid implementations), we set the parameters as follows: (a) The contraction factor is set to 1/3 rather than the standard 1/2 so that it better mimics DIRECT's pattern of points. (b) The step tolerance (which controls when GSS stops refining its step length and causes the method to exit when it is running alone or wait until a better external point is found) is set to 10^{-4}. (c) The sufficient decrease parameter was set to 10^{-8}. (d) The trial points at each iteration are ordered randomly before being submitted to the queue. In DIRECT, we set $\epsilon = 10^{-4}$ and limited the number of iterations to 10,000. In LHS, we set the number of trial points to 50. We limited the overall number of function evaluations to 15,000, and we set the cache comparison tolerance to 10^{-5}.

For each combination of problem, method, and number of processes, we ran five trials and report average results. GSS generally requires a different number of function evaluations for every run because it is asynchronous. DIRECT, on the other hand, is deterministic and always uses the same number of evaluations for the same problem, although the overall program does stop as soon as it finds a point which satisfies the stopping criteria — this means that it may stop in the middle of a set of DIRECT evaluations.

Figure 5 shows results on Branin [4, 36]. In this case, every trial was successful. Looking first at the non-hybrid methods, we see that GSS requires roughly 1/2 the function evaluations of DIRECT and 1/6 of the time; the reason for this disparity is that asynchronous GSS is better load-balanced. Neither method realizes any benefit by increasing the number of processors. GSS never has more than $2n$ evaluations running simultaneously and so can only use 5 processors (4 workers and one master). Note also that, even though GSS is a local method, it is able to find the global solution using fewer function evaluations and less time than DIRECT. In fact, GSS can be considered as an alternative to DIRECT because it is able to solve many problems that DIRECT can solve and often much more quickly. This is our motivation for combining them.

The hybrid methods combine the best features of these methods. The hybrid GD method is fastest overall and also uses the fewest function evaluations. Observe that the hybrid LGD method gets the most benefit from adding more processors. It uses nearly as many function evaluations as DIRECT but only requires 1/3 to 1/6 as much time. Moreover, there is near-perfect scaling from 8 to 16 processors, though very little benefit is realized from 16 to 32 because the problem is so small. Figure 5(b) shows the breakdown of evaluations by citizen within the hybrid method. For both hybrid methods, GSS used the most evaluations.

Figure 6 shows the results on the Six-Hump Camel problem [63], and every trial was successful. Once again, DIRECT is the most expensive method in terms of both the number of function evaluations and runtime. GSS and the hybrid GD method are nearly equivalent in runtime and the fastest overall, approximately ten times faster than DIRECT. The hybrid LGD is about the same speed for 16 or 32 processors.

For the first two problems, GSS was arguably the best method. On Goldstein

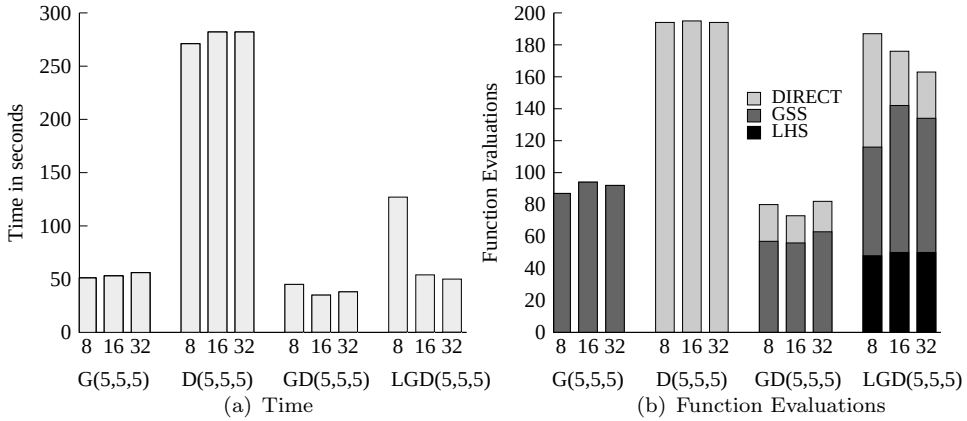


Figure 5. Average results over five runs on **Branin** ($n = 2$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

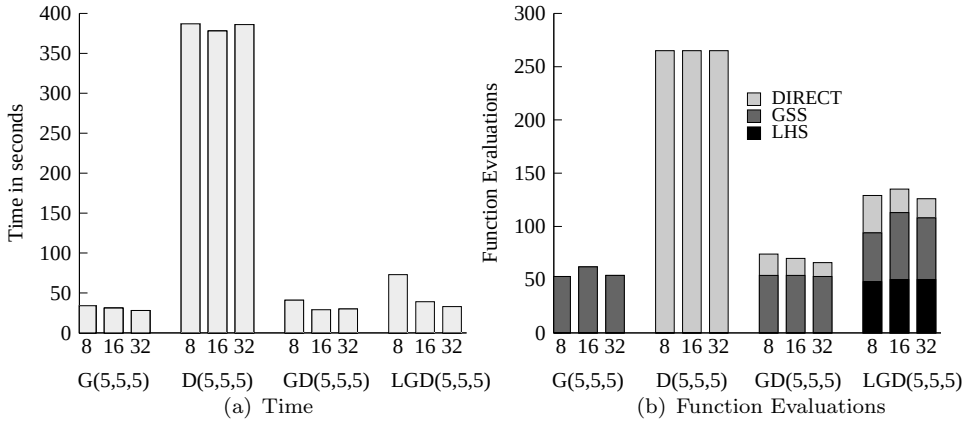


Figure 6. Average results over five runs on **Six-Hump Camel** ($n = 2$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

and Price [11] (see results in Figure 7) we see the problem with using GSS alone because GSS failed on some of the trials. So, even though it was fastest overall when it was able to solve the problem, it is not robust. Both hybrid methods are robust, though, and require only 1/5 of the running time of DIRECT.

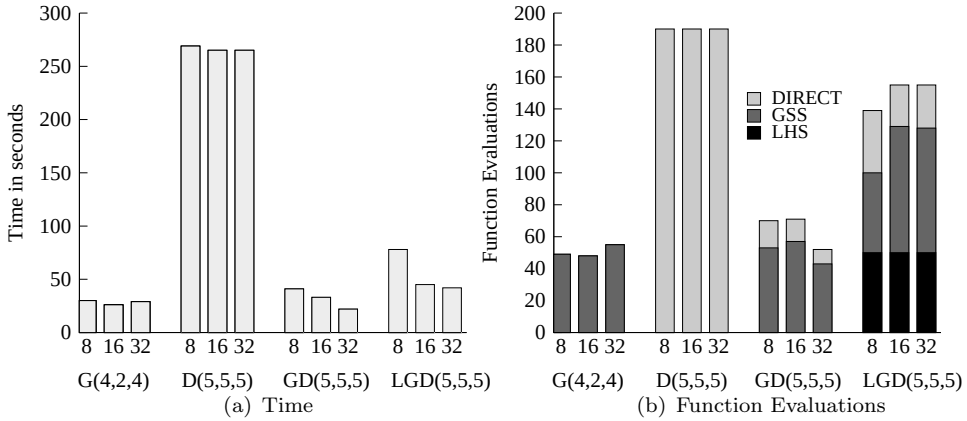
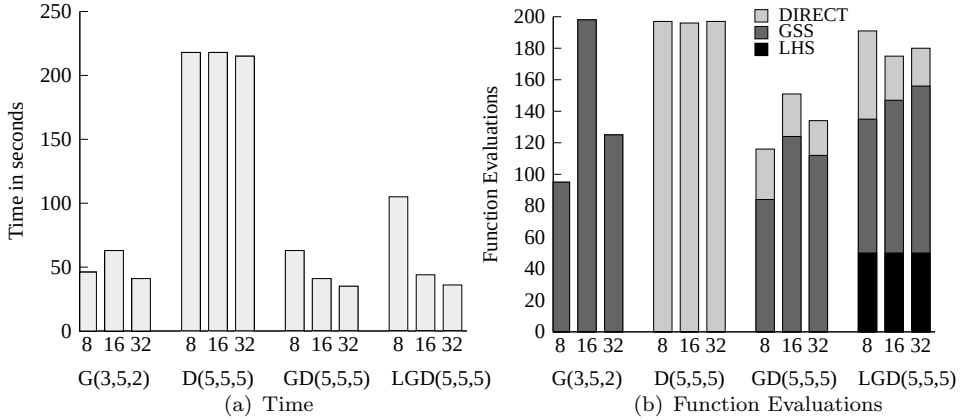


Figure 7. Average results over five runs on **Goldstein and Price** ($n = 2$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

The results for Hartman 3 [11], in Figure 8, are similar to those of the previous

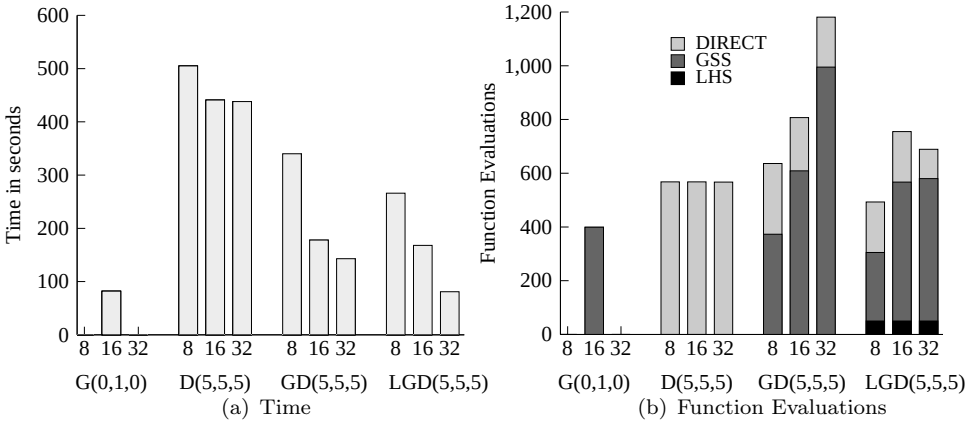
problem. GSS cannot always find the global solution. The hybrid methods can, however, and in less than 1/2 the time required by DIRECT.



(a) Time (b) Function Evaluations

Figure 8. Average results over five runs on **Hartman 3** ($n = 3$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

Thus far, we have not seen an advantage to adding an LHS citizen to our hybrid method. Generally, the hybrid LGD has usually been slower than the hybrid GD on 8 processors and about the same speed on 16 or 32 processors. On Hartman 6 [11], shown in Figure 9, we see that there can be a major advantage to the LHS sampling. This is a problem that requires 1,200 function evaluations in the worst case. The addition of the LHS sampling on some runs assisted GSS in quickly finding the global minimum. Also, as this is a problem with $n = 6$ variables, the benefits of scaling up the number of processors are more obvious. We also note that GSS could only solve this problem in one of 15 trials.



(a) Time (b) Function Evaluations

Figure 9. Average results over five runs on **Hartman 6** ($n = 6$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

On the Shekel problems [11], shown in Figures 10, 11, and 12, we see that GSS once again fails in many of the trials and DIRECT always requires the fewest overall function evaluations. Despite this seeming disadvantage, though, the hybrid methods are always faster. For example, consider Shekel 5 on 32 processors (see Figure 10). The hybrid LGD requires more than twice the number of evaluations as DIRECT but is about four times faster.

Shubert (see Figure 13) [63] is a problem that is well known to be difficult for DIRECT and requires 3,000 function evaluations to solve. GSS cannot solve the problem at all. The hybrid GD method for the most part does DIRECT iterations, as can be seen in Figure 13(b), and requires more evaluations overall than DIRECT,

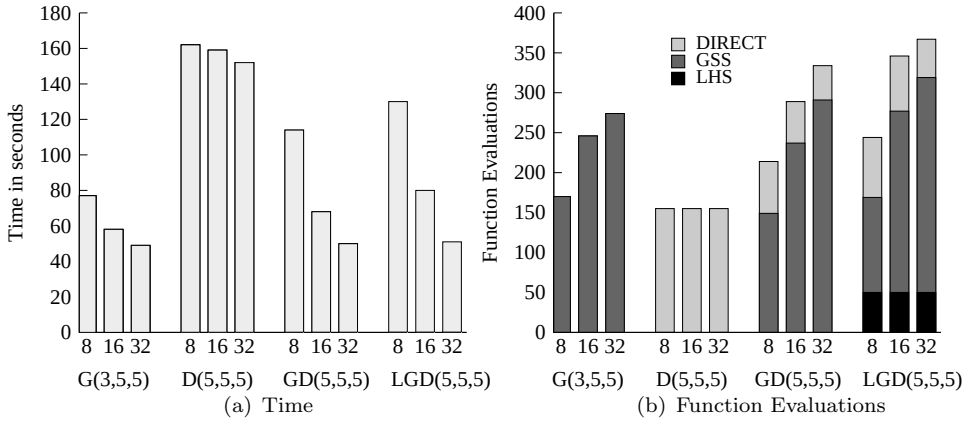


Figure 10. Average results over five runs on **Shekel 5** ($n = 4$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

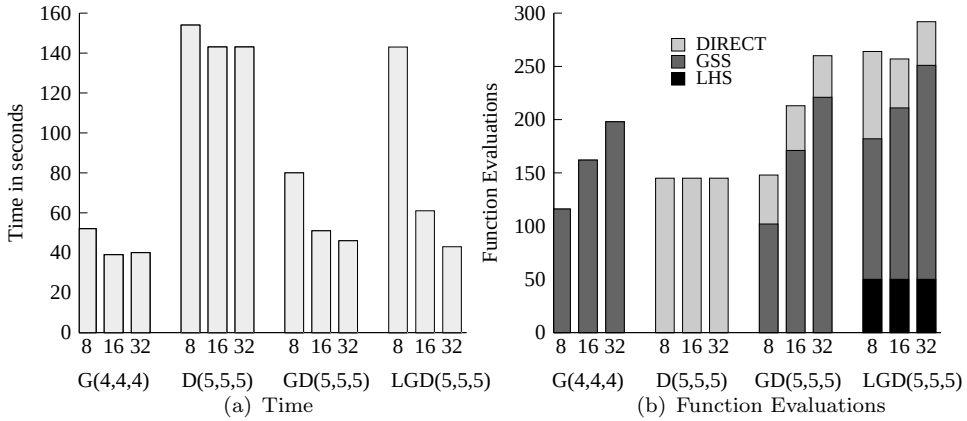


Figure 11. Average results over five runs on **Shekel 7** ($n = 4$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

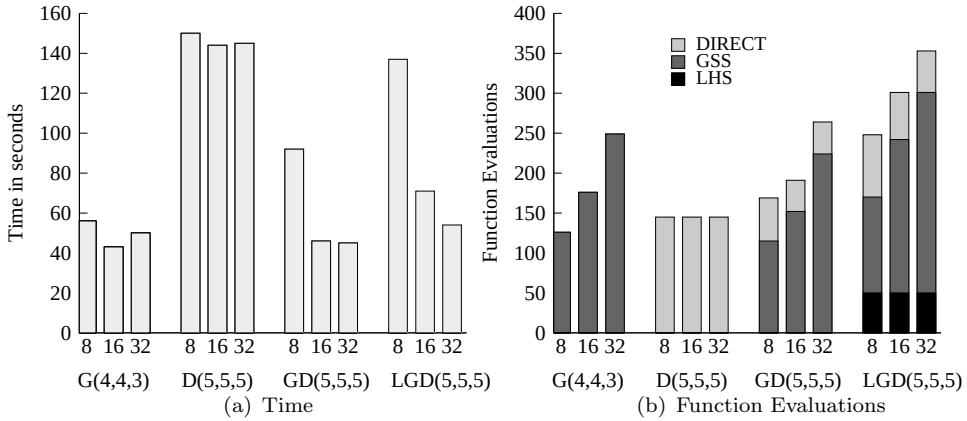
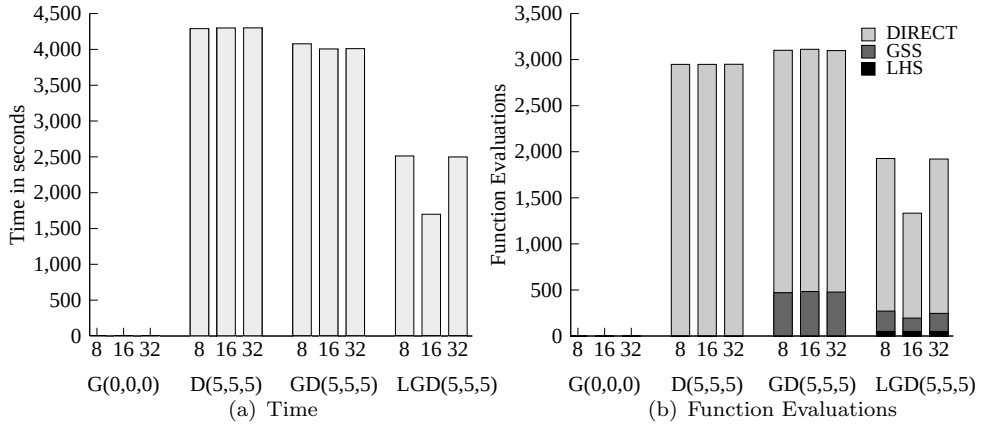


Figure 12. Average results over five runs on **Shekel 10** ($n = 4$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

but it is still slightly faster than DIRECT. The hybrid LGD still requires a large number of DIRECT evaluations but overall requires the fewest evaluations and is the fastest. In this case, we see little to no speed-up in going to a greater number of processors because DIRECT is the only citizen working for most of the time. GSS will stop running when it is locally converged, waiting for DIRECT or LHS to find a better solution than the best it has found so far.

Overall, both GSS and the hybrid methods are much faster in parallel than



(a) Time (b) Function Evaluations

Figure 13. Average results over five runs on **Shubert** ($n = 2$) for 8, 16, and 32 processors. The values in parentheses indicate how many trials were solved successfully for 8, 16, and 32 processors, respectively.

DIRECT, even when they require more function evaluations. This is because these other methods have better load balancing. Unlike GSS, which is asynchronous, DIRECT submits its trial points in batches. The next iteration of DIRECT cannot start until all points in the last batch have been evaluated. Suppose for example, that DIRECT submits 9 trial points. If only 8 evaluation processors are available each function evaluation takes 1 hour, then 2 hours will be required to evaluate all 9 points with 7 processors sitting idle during the second hour.

GSS, on the other hand, is an asynchronous method that was designed for good load balancing. Its iterations proceed as soon as one or more points have been evaluated. However, its current implementation can only scale to approximately $2n+1$ processors. Past that point, extra worker processors will sit idle. GSS is able to solve some of the global optimization problems, but is not a good alternative to DIRECT in general because it is not robust. One solution that has been used in the past is to start GSS at many different points; see [24].

The hybrid methods combine the positive features of GSS and DIRECT. The hybrid methods have good load balancing because they mix LHS, GSS, and DIRECT evaluations together. Our numerical results demonstrate runtimes that are always faster than DIRECT and generally as fast as GSS. These methods are also robust and successfully completed every trial in our example set. If enough processors are available, the hybrid LGD is generally to be preferred over the hybrid GD because of its robustness and speed. We have observed on other problems that the extra acceleration from LHS can be substantial, as in Figure 9. That said, most of the time the hybrid GD is the fastest method, especially when the number of processors is small.

6. Conclusions

In this paper, we have explored the benefits of parallel hybrid optimization combining GSS, DIRECT, and LHS. In addition to the motivations provided in the introduction, a major incentive for this work came from users of APPSPACK (which implements asynchronous GSS), who reported using GSS for global optimization problems. Although they were interested in finding a global optimum, they wanted a code that was as fast and convenient as their favorite local solver. The hybrid methods presented here are viable options for these users because the results in §5 suggest that they run as fast as GSS but are as robust as DIRECT and can, furhtermore, use more processors than either GSS or DIRECT alone. The hybrid

method has the same convergence properties as DIRECT (in fact, DIRECT is unmodified but simply runs alongside other methods).

Our hybrid scheme was a relatively simple one, but it is possible to consider many more options. For example, we could run multiple variants of DIRECT and GSS with different options. In our problems, we found that a step length reduction factor of 1/2 in GSS was better on some problems than 1/3. We could easily run two copies of GSS, one with each option. Likewise, there are several parameters in DIRECT that could be modified, especially the ϵ used in the determination of the potentially optimal rectangles.

Additionally, the algorithmic and software framework of HOPS described in §3 can be used with any solver citizen. The HOPS code is object-oriented and provides a generic interface for citizens. Previously, we have considered the problem of modifying DIRECT to use function evaluations from other citizens [22]. Other work [25, 26, 58] uses an early version of the HOPS framework to combine GSS, Treed Gaussian Process models (TGP), and LHS.

Acknowledgments

We gratefully acknowledge the users of APPSPACK for presenting us with new and challenging scenarios that motivated us to extend the code and create the new HOPSPACK framework presented here. We are indebted to Genetha Gray for her input on the design of HOPSPACK. Noam Goldberg was kind enough to proofread sections of this manuscript. Finally, we thank the two anonymous referees for their valuable suggestions and also Mihai Antiescu for his handling of the manuscript.

References

[1] E. Alba, editor. *Parallel Metaheuristics*. John Wiley & Sons, Inc., 2005.

[2] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2):299–318, June 2008.

[3] M. C. Bartholomew-Biggs, S. C. Parkhurst, and S. R. Wilson. Global optimization — stochastic or deterministic? In A. Albrecht and K. Steinhofel, editors, *SAGA 2003: 2nd International Symposium on Stochastic Algorithms*, volume 2827 of *Lecture Notes in Computer Science*, pages 125 – 137. Springer-Verlag, 2003.

[4] M. Björkman and K. Holmström. Global optimization using the DIRECT algorithm in Matlab. *Advanced Modeling and Optimization*, 2(2):17–37, 1999.

[5] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17(1):1–13, Feb. 1999.

[6] J. P. Castro, G. A. Gray, A. A. Giunta, and P. D. Hough. Developing a computationally efficient dynamic multilevel hybrid optimization scheme using multifidelity model interactions. Technical Report SAND2005-7498, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, Nov. 2005.

[7] S. I. Chernyshenko and A. V. Privalov. Internal degrees of freedom of an actuator disk model. *Journal of Propulsion and Power*, 20(1):155–163, Jan.–Feb. 2004.

[8] M. L. Chiesa, R. E. Jones, K. J. Perano, and T. G. Kolda. Parallel optimization of forging processes for optimal material properties. In *NUMIFORM 2004: The 8th International Conference on Numerical Methods in Industrial Forming Processes*, volume 712 of *AIP Conference Proceedings*, pages 2080–2084, 2004.

[9] L. Chiter. DIRECT algorithm: A new definition of potentially optimal hyperrectangles. *Applied Mathematics and Computation*, 179(2):742–749, 2006.

[10] S. E. Cox, W. E. Hart, R. Haftka, and L. Watson. . In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002. AIAA-2002-5581.

[11] L. C. W. Dixon and G. P. Szego. The global optimization problem: An introduction. In *Towards Global Optimization*, pages 1–15. North-Holland Publishing Company, 1978.

[12] T. Eitrich and B. Lang. Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of Computational and Applied Mathematics*, 196:425–436, 2006.

[13] S.-K. S. Fan and E. Zahara. A hybrid simplex search and particle swarm optimization for unconstrained optimization. *European Journal of Operational Research*, 181(2):527–548, 2007.

[14] K.-T. Fang, R. Li, and A. Sudjianto. *Design and Modeling for Computer Experiments (Computer Science & Data Analysis)*. Chapman & Hall/CRC, 2005.

[15] D. E. Finkel and C. T. Kelley. An adaptive restart implementation of DIRECT. Technical Report CRSC-TR04-30, Center for Research in Scientific Computation, North Carolina State University, August 2004.

[16] D. E. Finkel and C. T. Kelley. Convergence analysis of the DIRECT algorithm. Technical Report CRSC-TR04-28, Center for Research in Scientific Computation, North Carolina State University, 2004.

[17] D. E. Finkel and C. T. Kelley. Additive scaling and the DIRECT algorithm. *Journal of Global Optimization*, 36(4):597–608, 2006.

[18] 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, J. M. Gablonsky, G. Gray, and T. G. Kolda. A comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, May 2008.

[19] J. Gablonsky. Fortran code for DIRECT. `http://www4.ncsu.edu/~ctk/SOFTWARE/DIRECTv204.tar.gz`, 2001.

[20] J. M. Gablonsky. *Modifications of the DIRECT Algorithm*. PhD thesis, North Carolina State University, 2001.

[21] J. M. Gablonsky and C. T. Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21(1):27–37, Sept. 2001.

[22] N. Goldberg, T. G. Kolda, and A. S. Yoshimura. Concurrent optimization with DUET: DIRECT using external trial points. Technical Report SAND2008-5844, Sandia National Laboratories, Livermore, CA, 2008.

[23] G. A. Gray and T. 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.

[24] G. A. Gray, T. G. Kolda, K. L. Sale, and M. M. Young. Optimizing an empirical scoring function for transmembrane protein structure determination. *INFORMS Journal on Computing*, 16(4):406–418, 2004. Special Issue on Computational Molecular Biology/Bioinformatics.

[25] G. A. Gray, M. Taddy, J. D. Griffin, M. Martinez-Canales, and H. K. H. Lee. Hybrid optimization: A tool for model calibration. in preparation, 2008.

[26] G. A. Gray, M. Taddy, M. Martinez-Canales, and H. K. H. Lee. Enhancing parallel pattern search optimization with a Gaussian process oracle. In *Proceedings of the 14th Nuclear Explosive Codes Development Conference (NECDC)*, 2007.

[27] J. D. Griffin and T. G. Kolda. Nonlinearly-constrained optimization using asynchronous parallel generating set search. Technical Report SAND2007-3257, Sandia National Laboratories, May 2007.

[28] J. D. Griffin, T. G. Kolda, and R. M. Lewis. Asynchronous parallel generating set search for linearly-constrained optimization. Technical Report SAND2006-4621, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, Aug. 2006.

[29] J. C. Helton and F. J. Davis. Sampling-based methods for uncertainty and sensitivity analysis. Technical Report SAND99-2240, Sandia National Laboratories, Albuquerque, NM, July 2000.

[30] C. Hernández. *Stereo and Silhouette Fusion for 3D Object Modeling from Uncalibrated Images Under Circular Motion*. PhD thesis, Ecole Nationale Supŕieure des Télécommunications, France, May 2004.

[31] C. Hogea, C. Davatzikos, and G. Biros. An image-driven parameter estimation problem for a reaction-diffusion glioma growth model with mass effects. *Journal of Mathematical Biology*, 56(6):793–825, June 2008.

[32] P. D. Hough, T. G. Kolda, and H. A. Patrick. Usage manual for APPSPACK 2.0. Technical Report SAND2000-8843, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, 2000.

[33] P. D. Hough, T. G. Kolda, and V. J. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, June 2001.

[34] R. Iman and W. Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics*, 11(3):311–334, 1982.

[35] D. R. Jones. Direct global optimization algorithm. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 431–440. Kluwer Academic Publishers, 2001.

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

[37] T. G. Kolda. Revisiting asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Optimization*, 16(2):563–586, Dec. 2005.

[38] 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, Aug. 2003.

[39] T. G. Kolda, R. M. Lewis, and V. Torczon. A generating set direct search augmented Lagrangian algorithm for optimization with a combination of general and linear constraints. Technical Report SAND2006-5315, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, Aug. 2006.

[40] T. G. Kolda, R. M. Lewis, and V. Torczon. Stationarity results for generating set search for linearly constrained optimization. *SIAM Journal on Optimization*, 17(4):943–968, 2006.

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

[42] M. A. Kupinski, E. Clarkson, J. W. Hoppin, L. Chen, and H. H. Barrett. Experimental determination of object statistics from noisy images. *Journal of the Optical Society of America A*, 20(3):421–429, March 2003.

[43] C. Lakhdar. Towards a new DIRECT algorithm: a two-points based sampling method. Department of Mathematics, University Ferhat-Abbas, Algeria, `http://www.optimization-online.org/DB_FILE/2005/03/1077.pdf`, 2005.

[44] R. M. Lewis, A. Shepherd, and V. Torczon. Implementing generating set search methods for linearly constrained minimization. *SIAM Journal on Scientific Computing*, 29(6):2507–2530, 2007.

[45] R. M. Lewis and V. Torczon. A globally convergent augmented Lagrangian pattern search algo-

rithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002.

[46] J. Liang and Y.-Q. Chen. Optimization of a fed-batch fermentation process control competition problem using the NEOS server. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 217(5):427–342, March 2003.

[47] 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.

[48] D. McKee. A dynamic model of retirement in Indonesia. Technical Report CCPR-005-06, California Center for Population Research On-Line Working Paper Series, USA, Feb. 2006.

[49] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. In *Acta Numerica 2004*. Cambridge University Press, 2004.

[50] S. O. Nielsen, C. F. Lopez, G. Srinivas, and M. L. Klein. Coarse grain models and the computer simulation of soft materials. *Journal of Physics: Condensed Matter*, 16:R481–R512, 2 Apr. 2004.

[51] J. L. Payne and M. J. Eppstein. A hybrid genetic algorithm with pattern search for finding heavy atoms in protein crystals. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 377–384, New York, NY, USA, 2005. ACM.

[52] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):1436–4646, May 2002.

[53] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *IWINAC 2005: Proceedings of First International Workshop on the Interplay between Natural and Artificial Computation*, pages 41–53, 2005.

[54] G. R. Raidl. A unified view on hybrid metaheuristics. In *HM06: Third International Workshop on Hybrid Metaheuristics*, pages 1–12, 2006.

[55] R. G. Regis and C. A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31(1):153–171, Jan. 2005.

[56] T. J. Santer, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer Verlag Publication, New York, USA, 2003.

[57] E. S. Siah, M. Sasena, J. L. Volakis, P. Y. Papalambros, and R. W. Wiese. Fast parameter optimization of large-scale electromagnetic objects using DIRECT with Kriging metamodeling. *IEEE Transactions on Microwave Theory and Techniques*, 52(1):276–285, Jan. 2004.

[58] M. Taddy, H. K. H. Lee, G. A. Gray, and J. D. Griffin. Bayesian guided pattern search for robust local optimization. Submitted to Technometrics, Jan. 2008.

[59] A. I. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, 39(2):197–219, 2007.

[60] K. P. Wachowiak and T. M. Peters. Combining global and local parallel optimization for medical image registration. In J. Fitzpatrick, JM; Reinhardt, editor, *Medical Imaging 2005: Image Processing*, volume 5747, pages 1189–1200. SPIE, Apr. 2005.

[61] M. P. Wachowiak and T. M. Peters. Parallel optimization approaches for medical image registration. In *MICCAI 2004: 7th International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 3216 of *Lecture Notes in Computer Science*, pages 781–788. Springer-Verlag, 2004.

[62] G. D. Wyss and K. H. Jorgensen. A user's guide to LHS: Sandia's Latin hypercube sampling software. Technical Report SAND98-0210, Sandia National Laboratories, Albuquerque, NM, February 1998.

[63] Y. Yao. Dynamic tunneling algorithm for global optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1222–1230, 1989.

[64] P. Yehui and L. Zhenhai. A derivative-free algorithm for unconstrained optimization. *Applied Mathematics — A Journal of Chinese Universities*, 20(4):491–498, December 2007.

[65] E. I. Zacharaki, C. Hogea, G. D. Shen, G. Biros, and C. Davatzikos. Parallel optimization of tumor model parameters for fast registration of brain tumor images. In *Proc. SPIE*, volume 6914, 2008.

[66] T. Zhang, K. K. Choi, S. Rahman, K. Cho, P. Baker, M. Shakil, and D. Heitkamp. A hybrid surrogate and pattern search optimization method and application to microelectronics. *Structural and Multidisciplinary Optimization*, 32(4):327–345, October 2006.

Appendix A. Test problem descriptions

We use the test problems from the original DIRECT experiments in [36]. Table A1 summarizes the characteristics of the problems in this group [4].

Table A1. Summary of test problems.

Problem	**Abbrev.**	**Problem Size**	**# Local Minima**	**# Global Minima**	**Global Minimum**
Branin	BRA	2	3	3	0.397887
Six-Hump Camel	CA6	2	6	2	-1.03163
Goldstein and Price	GOL	2	6	2	3.00000
Hartman 3	HA3	3	4	1	-3.86278
Hartman 6	HA6	6	4	1	-3.32237
Shekel 5	SH5	4	5	1	-10.1532
Shekel 7	SH7	4	7	1	-10.4029
Shekel 10	SH10	4	10	1	-10.5364
2D Shubert	SHU	2	760	18	-186.731

Branin (BRA). There are two versions of the two-dimensional Branin test function. The one listed in [4] is

$$f(x) = \left(x_2 - \frac{5x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10.$$

However, others have used

$$f(x) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10.$$

In our informal experiments, the first version matches the results in [4], while the second matches the results in [36] (which does not explicitly list the function). We use the second version (with 5.1) in our results. Both have constraints

$$-5 \le x_1 \le 10, \quad 0 \le x_2 \le 15.$$

Six-Hump Camel (CA6). The two-dimensional six-hump camel test function [63] is

$$f(x) = \left(4 - 2.1\,x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2,$$

with constraints

$$-3 \le x_1 \le 3, \quad -2 \le x_2 \le 2.$$

Goldstein-Price problem (GOL). The two-dimensional Goldstein-Price test function [11] is

$$f(x) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \cdot \left[30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right].$$

with constraints

$$-2 \le x_i \le 2 \text{ for } i = 1, 2.$$

Hartman (HA3/HA6). The two Hartman test problems [11] are defined for $n = 3$ and $n = 6$ variables by

$$f(x) = -\sum_{j=1}^{4} c_j \exp\left(-\sum_{i=1}^{n} a_{ij}(x_i - p_{ij})^2\right),$$

with constraints

$$0 \le x_i \le 1 \text{ for } i = 1, \ldots, n.$$

For both $n = 3$ and $n = 6$, we have

$$c = \left[1.0\ 1.2\ 3.0\ 3.2\right].$$

For $n = 3$, the matrices A and P are

$$A = \begin{bmatrix} 3.0 & 0.1 & 3.0 & 0.1 \\ 10.0 & 10.0 & 10.0 & 10.0 \\ 30.0 & 35.0 & 30.0 & 35.0 \end{bmatrix} \text{ and } P = \begin{bmatrix} 0.36890 & 0.46990 & 0.10910 & 0.03815 \\ 0.11700 & 0.43870 & 0.87320 & 0.57430 \\ 0.26730 & 0.74700 & 0.55470 & 0.88280 \end{bmatrix}.$$

For $n = 6$, the matrices A and P are

$$A = \begin{bmatrix} 10.00 & 0.05 & 3.00 & 17.00 \\ 3.00 & 10.00 & 3.50 & 8.00 \\ 17.00 & 17.00 & 1.70 & 0.05 \\ 3.50 & 0.10 & 10.00 & 10.00 \\ 1.70 & 8.00 & 17.00 & 0.10 \\ 8.00 & 14.00 & 8.00 & 14.00 \end{bmatrix} \text{ and } P = \begin{bmatrix} 0.1312 & 0.2329 & 0.2348 & 0.4047 \\ 0.1696 & 0.4135 & 0.1451 & 0.8828 \\ 0.5569 & 0.8307 & 0.3522 & 0.8732 \\ 0.0124 & 0.3736 & 0.2883 & 0.5743 \\ 0.8283 & 0.1004 & 0.3047 & 0.1091 \\ 0.5886 & 0.9991 & 0.6650 & 0.0381 \end{bmatrix}.$$

Shekel (SH5/SH7/SH10). The three four-dimensional Shekel test functions [11] are defined by

$$f(x) = -\sum_{j=1}^{m} \left(c_j + \sum_{i=1}^{4} (x_i - a_{ij})^2 \right)^{-1},$$

for $m = 5, 7, 10$ with constraints

$$0 \le x_i \le 10 \text{ for } i = 1, 2, 3, 4.$$

Here

$$c = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.3 & 0.7 & 0.5 & 0.5 \end{bmatrix}$$

and

$$A = \begin{bmatrix} 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 5 & 1 & 2 & 3.6 \\ 4 & 1 & 8 & 6 & 3 & 2 & 3 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3.6 \end{bmatrix}.$$

Shubert (SHU). The two-dimensional Shubert problem [63] is

$$f(x) = \left(\sum_{i=1}^{5} i \cos\left((i+1)x_1 + i\right) \right) \left(\sum_{i=1}^{5} i \cos\left((i+1)x_2 + i\right) \right),$$

with constraints

$$-10 \le x_i \le 10 \text{ for } i = 1, 2.$$