14th INFORMS Computing Society Conference
Richmond, Virginia, January 11–13, 2015
pp. 47–61

http://dx.doi.org/10.1287/ics.2015.0004

A Dividing Rectangles Algorithm for Stochastic Simulation Optimization

Paul Nicholas
Operations Analysis Division, U.S. Marine Corps, Quantico, Virginia 22134, USA,
paul.nicholas@usmc.mil

Abstract Many important real-world optimization problems are stochastic, non-linear, non-differentiable, and non-convex. To solve these difficult problems, practitioners often rely on black-box optimization methods which make no or few assumptions of internal problem structure. The Dividing Rectangles (DIRECT) algorithm has proven popular for deterministic, low-dimensional black-box problems. The algorithm does not require calculation of a gradient, is relatively simple (having only one tuning parameter), and is widely applicable, requiring only that the function be Lipschitz continuous. We describe a new DIRECT algorithm to support stochastic simulation optimization that retains the global convergence properties of the original. Our algorithm iterates between searching for new solutions and reducing the variance of existing solutions using Bayesian sample information. The algorithm intelligently allocates available computational resources using the Optimal Computing Budget Allocation (OCBA) methodology, and includes several heuristic mechanisms which in practice increase the likelihood of quickly obtaining desirable solutions. In computational experiments, we find this new algorithm can provide better performance than the only other known modification of DIRECT for stochastic problems. We provide the results of a realistic application where we use the algorithm to design a wireless mesh network supporting uncertain traffic demands.

Keywords Dividing Rectangles; DIRECT; stochastic optimization; simulation optimization; direct search; optimal computing budget allocation; OCBA

1. Introduction

Many real-world optimization problems have no analytically tractable closed-form solution and can only be calculated using simulation. Applications include determining optimal vehicular traffic management policies, finding ideal characteristics for a weapons platform within a high-fidelity combat simulation, and designing a wireless communications network to best support unpredictable customer demand. These problems may be stochastic, non-linear, non-differentiable, and non-convex. We consider problems of the form:

$$\min_{x\in\Omega} f(x) = E[F(x,\omega)] \tag{1}$$

where the domain $\Omega = \{x \in \mathbb{R}^n : l \le x \le u\}$, l and u are the respective lower and upper bounds for each variable x, and ω represents the randomness or noise present in the function. We are unable to exactly compute the underlying function $f(\cdot)$, and so we must estimate using the computable sample performance estimate $F(\cdot,\omega)$.

Such problems can be difficult to solve for several reasons. First, the lack of differentiability and convexity removes the possibility of using exact gradient methods to find good solutions and prove optimality; thus one must often settle for "good enough" solutions. Second, running the simulation itself may be computationally intensive, and there may only be a finite computational budget available. As Fu et al. [16] poignantly note, "The key difficulty in [stochastic] simulation optimization involves a trade-off between allocating computational resources for searching the solution space versus conducting additional function evaluations for better estimating the performance of current promising solutions. . . The fundamental tradeoff between search and estimation becomes especially pronounced when the cost of simulation is expensive." In contrast, deterministic methods may focus computational effort entirely on search, as the value of each solution – once calculated via simulation or other means – is known with certainty.

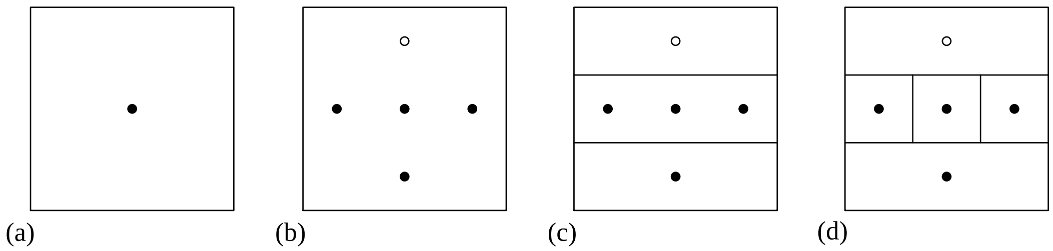
Because of these difficulties, practitioners often rely on *black-box optimization* methods to obtain solutions. These methods require little or no knowledge of the structure of the underlying problem, and generally have few tuning parameters, making them popular for real-world applications. The *Dividing Rectangles* (*DIRECT*) algorithm of Jones et al. [29] is a popular black-box optimization method used for deterministic, low-dimensional problems. DIRECT is attractive because it has proven global convergence and only one tuning parameter, to which results are fairly robust. The only requirement of DIRECT is that the function be *Lipschitz continuous*, i.e., the maximum rate of change of the function must be bounded by a finite real constant. This requirement does not imply the function must be linear, convex, or even differentiable (Jeffreys and Jeffreys [27]).

As of November 2014, Google Scholar lists 1,140 citations of the original DIRECT article by Jones et al. [29]. DIRECT has been applied to many large-scale industrial problems, including Zhu and Bogy [52], He et al. [21], Saber and Shaw [42], and Nicholas and Alderson [39, 40]. While DIRECT is formulated to support deterministic optimization, it has been successfully applied to stochastic problems with little or no modification. Di Serafino et al. [13] use information obtained from a grid search to inform the progress of DIRECT in detecting gravitational waves from coalescing binary star systems, ignoring the Gaussian noise present in their objective function. Wachowiak and Peters [48] and He and Narayana [22] use DIRECT for medical image registration in the presence of noise. Hemker and Werner [23] develop a deterministic DIRECT method that performs well on noisy functions. Deng [11] and Deng and Ferris [12] develop the only other known variation of DIRECT that is specifically modified to support stochastic simulation optimization. They show their *Noisy DIRECT* algorithm (also used by Esmaeilzadeh Azar and Perrier [4]) performs better than deterministic DIRECT on a noisy function.

We develop a new stochastic version of DIRECT, called *DIRECT-S*, to search for good solutions to stochastic simulation optimization problems while efficiently allocating a limited computational budget. Our algorithm iterates between a *search* phase (wherein we use function evaluations to further explore the solution space) and a *refinement* phase (wherein we use function evaluations to update Bayesian posterior distributions of existing solutions). To improve the efficiency of our refinement phase, we incorporate the *Optimal Computing Budget Allocation* (*OCBA*) method of Chen ([7, 8]). In numerical testing, we find our algorithm can provide better performance than the black-box optimization method of Anderson and Ferris [2] and both a naïve implementation of deterministic DIRECT and the Noisy DIRECT algorithm of Deng [11] and Deng and Ferris [12]. Our algorithm is relatively simple to implement and maintains the convergence properties of the original, deterministic version of DIRECT.

This paper is organized as follows. In the next section, we briefly describe the original, deterministic DIRECT algorithm. In Section 3 we describe our DIRECT-S algorithm in detail. In Section 4 we provide computational results using benchmark problems and compare our algorithm to other methods. In Section 5 we use our algorithm to design a realistic wireless mesh network topology to support probabilistic customer traffic demand. We conclude in Section 6 with recommendations for future work.

FIGURE 1. DIRECT sampling and division.



Note. DIRECT first samples from the center of the solution space (a) and along one-third of each dimension (b). It then divides the solution space by placing the best solution (open circle) into the largest new hyper-rectangle (c-d).

2. Original Dividing Rectangles (DIRECT)

We provide an overview of the original deterministic DIRECT algorithm based largely on Nicholas [38]; the interested reader is referred to Jones et al. [29], Finkel [14], and Finkel and Kelley [15] for more details. The algorithm is initialized by normalizing the original hyper-rectangular solution space Ω to create a unit hyper-cube. Consider a simple two-dimensional example in Figure 1. From this hyper-cube (in this case, a square representing the entire two-dimensional solution space), the algorithm will iteratively sample selected hyper-rectangles and then divide them into smaller hyper-rectangles. In the first iteration, it will first sample directly in the middle of Ω (Figure 1(a)). Let M be the set of all undivided hyper-rectangles, indexed by $i = 1, 2, \ldots, m$ (alias j). Let $f(\mathbf{c}_i)$ denote the function value of a particular hyper-rectangle i at center point $\mathbf{c}_i$, and let d_i denote the distance from the center point to the corner of the hyper-rectangle, i.e., the *size* of the hyper-rectangle. The algorithm samples the original hyper-rectangle at $\mathbf{c}_i \pm \delta \mathbf{e}_r$, where δ is one-third the side-length of the largest dimension, and $\mathbf{e}_r$ is the r^{th} unit vector; see Figure 1(b).

The algorithm then divides the solution space into smaller hyper-rectangles, hence the name *DIviding RECTangles*. For each newly-sampled point, it places the best point (indicated as an open circle in Figure 1) into the largest new sub-hyper-rectangle by first dividing the original hyper-rectangle along the dimension with the best objective value (in Figure 1(c), the vertical dimension). The process continues for the dimension with the next best objective value (in Figure 1(d), the horizontal dimension).

2.1. Selecting Potentially Optimal Hyper-Rectangles

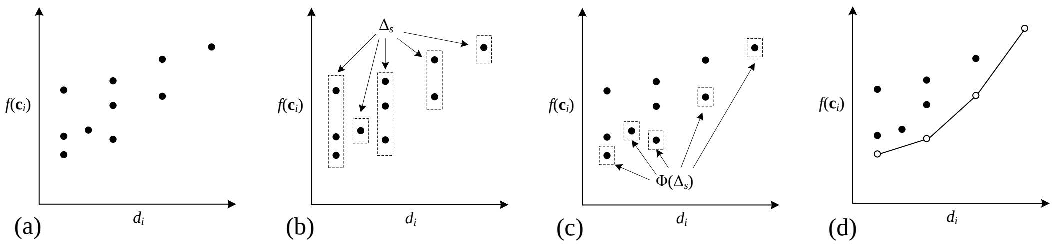
The key element of DIRECT is determining those hyper-rectangles to sample. If the algorithm always sampled all sub-hyper-rectangles, the search would forever remain global and would never focus on a particular local optimum. To prevent this, the algorithm only samples from those sub-hyper-rectangles it determines are *potentially optimal.* To aid in defining this term, consider Figure 2 where the function objective values $f(\mathbf{c}_i)$ are plotted against d_i for various hyper-rectangles $i \in M$ (represented as dots). The vertical axis represents goodness with respect to *local* search: lower $f(\mathbf{c}_i)$ values are closer to the actual global minimum. The horizontal axis represents goodness with respect to *global* search: larger d_i values have more unexplored territory in their respective hyper-rectangles, and thus in general have more potential for improvement.

Let Δ_s be the set of all hyper-rectangles of the same size d, for $s = 1, 2, \ldots, S$. Thus each Δ_s contains all hyper-rectangles i with the same abscissa, i.e., the dashed boxes in Figure 2(b), for $i = 1, 2, \ldots, I_s$. Let $\Phi(\Delta_s)$ be the hyper-rectangle of Δ_s with the best objective value, that is:

$$\Phi(\Delta_s) = \{i : f(\mathbf{c}_i) \leq f(\mathbf{c}_j)\ \forall i, j \in I_s\} \tag{2}$$

depicted in Figure 2(c). Let Θ be the set of all $\Phi(\Delta_s)$, that is, $\Theta = \bigcup_{s \in S} \Phi(\Delta_s)$.

FIGURE 2. Determining potentially optimal hyper-rectangles.



Note. Hyper-rectangles plotted as a function of size d_i (horizontal axis) and objective value $f(\mathbf{c}_i)$ (vertical axis) (a). Each Δ_s contains those hyper-rectangles of the same size, i.e., within the same abscissa (b). The $\Phi(\Delta_s)$ of each Δ_s has the best (i.e., lowest) objective value (c). The set Π of potentially optimal hyper-rectangles are those points on the convex hull (open circles in (d)) that satisfy (3) and (4).

In modifying *Lipschitzian optimization* [24], Jones et al. [29] introduce a rate-of-change parameter $\tilde{k}$ to vary the scope of search between global and local. The $\tilde{k}$ value changes dynamically during the course of the algorithm to find the set Π of potentially optimal hyper-rectangles, for $i = 1, 2, \ldots, |\Pi|$. A hyper-rectangle i is defined to be potentially optimal if there exists some $\tilde{k} > 0$ such that

$$f(\mathbf{c}_i) - \tilde{k}d_i \le f(\mathbf{c}_j) - \tilde{k}d_j \quad \forall j = 1, 2, \ldots, m, \tag{3}$$

$$f(\mathbf{c}_i) - \tilde{k}d_i \le f(\mathbf{c}^*) - \epsilon|f(\mathbf{c}^*)| \tag{4}$$

where $\mathbf{c}$ and $f(\mathbf{c}^*)$ are respectively the center point and objective function value of the current best hyper-rectangle or *incumbent solution*, and ϵ is a small positive constant. The first equation ensures that only those hyper-rectangles on the lower-right border of the cloud of dots in Figure 2(d) (i.e., the *convex hull*) can be potentially optimal. The second equation ensures that the lower bound $f(\mathbf{c}_i) - \tilde{k}d_i$ of each potentially optimal hyper-rectangle falls at or below the lower bound of the current incumbent solution by a nontrivial amount; this prevents the search from becoming too local.

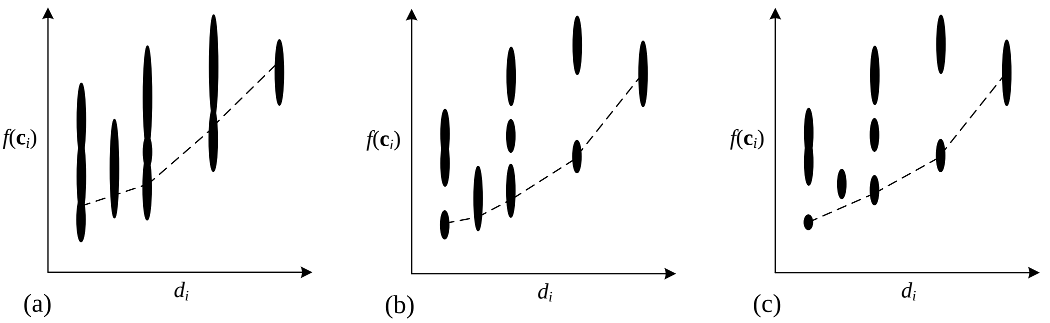
During each iteration, the algorithm will determine the set Π, sample each $i \in \Pi$, and then divide each $i \in \Pi$ into smaller hyper-rectangles. Since each hyper-rectangle i will become smaller and sampling depends on the size of each dimension, no point in the solution space is ever sampled twice. The algorithm will sample from at least one hyper-rectangle at each iteration, so it will never "run out" of available hyper-rectangles. Typical DIRECT stopping criteria include total function evaluations and DIRECT iterations.

2.2. Convergence

As Jones et al. [29] and Finkel and Kelley [15] show, the DIRECT algorithm is guaranteed to eventually converge to the global optimum for a deterministic, Lipschitz continuous function. Proof of this convergence relies on the selection of Π.

At each iteration, the $\Phi(\Delta_s)$ from the largest Δ_s (i.e., farthest to the right in Figure 4) will be added to Π. Since any hyper-rectangle added to Π must then be divided and placed into smaller Δ_s, and the total number of hyper-rectangles within any Δ_s are finite, each hyper-rectangle in the largest Δ_s will be sampled. As the algorithm progresses, every Δ_s will eventually be the largest, so every hyper-rectangle will be sampled as the number of iterations approaches infinity. At any step of the algorithm, the set of all hyper-rectangles precisely fill the original solution space, and the size of each hyper-rectangle necessarily gets smaller, creating an increasingly dense set of sampled points. Hence, the algorithm will eventually sample within an arbitrary distance of any point in the solution space, including the global optima. Note that despite this guaranteed convergence, the algorithm is unable to determine when it has achieved optimality unless it is provided additional information (such as a lower bound).

Figure 3. Refining the convex hull in the presence of noise.



Note. In the presence of noise, the points of Figure 1 become "blurs" due to the variance in each finite sample (a). In (b), we expend additional function evaluations to increase the probability of selecting $\Phi(\Delta_s)$ within each abscissa $s \in S$. In (c), we expend additional function evaluations to increase the probability of selecting the incumbent solution. These refinement actions reduce the size of each blur and may change the convex hull, thus also changing the set of potentially optimal hyper-rectangles Π.

3. DIRECT-S

When using the DIRECT algorithm for stochastic problems, the points present in Figure 1 are more accurately represented as collections of points or "blurs" (see Figure 3), as the true objective value of the center-point of each rectangle is subject to noise (we assume normally-distributed noise). We call the application of the DIRECT algorithm to a stochastic problem *naïve* if it simply calculates the objective function a fixed number of times for each sampled point, uses the sample mean as an estimate of the true value, and contains no other adaptation to account for stochasticity. In practice, the naïve method can perform well (see, e.g., [13, 22, 23, 48]).

As Deng [11] and Deng and Ferris [12] observe, the naïve method may incorrectly select the set Π, wasting computational effort searching undesirable portions of the solutions space. It may also incorrectly select the incumbent solution $\mathbf{c}^*$, as it has no mechanism for reducing variance and thus distinguishing between similar solutions. Their Noisy DIRECT algorithm avoids this problem by using Bayesian posterior distribution information to iteratively reduce the uncertainty of solution objective function values. They then conduct Monte Carlo simulation to create different versions of Π, and compare the elements in these sets to an original set. They increase the number of function evaluations until the randomly-sampled sets contain a large enough (i.e., 90%) portion of the original set. They find their method performs better than the naïve DIRECT method and the *Snobfit* (*Stable Noisy Optimization by Branch and Fit*) solver of Huyer and Neumaier [25] on several noisy functions.

Our approach similarly uses Bayesian sample information, but we do not conduct Monte Carlo simulation on the set Π. Rather, our algorithm adds several mechanisms wherein we expend additional function evaluations and use a filtering technique to improve the probability of correctly selecting Π. After conducting additional function evaluations during the refinement phase, the objective function values of each hyper-rectangle are known with better precision (i.e., less variance or smaller "blurs" in Figure 3(b)). This can change the convex hull and cause the addition or removal of hyper-rectangles from the set Π (see Figure 3(c)). We find that by refining Π in this manner, in practice we can better avoid exploring fruitless portions of the solution space and provide more desirable solutions in fewer function evaluations, even considering the additional function evaluations expended during refinement. The following sections explain our two refinement mechanisms and filtering mechanism in detail.

3.1. Refining Each Abscissa

Our first mechanism for improving the performance of DIRECT-S is designed to increase the probability of correctly selecting each $\Phi(\Delta_s)$, i.e., we wish to reduce the "blur" of each abscissa in Figure 3(a). In practice, this will increase the chances of correctly identifying those i satisfying (3), i.e., those hyper-rectangles on the convex hull. We compute the *approximate probability of correct selection* (*APCS*) for each $\Phi(\Delta_s)$, denoted $APCS(\Phi(\Delta_s))$, using the Bayesian model of Chen and Lee [7]. If this probability is below a user-defined threshold $\tau_{abscissa}$, we expend $T_{abscissa}$ total additional function evaluations, allocating N_i^s function evaluations to each Δ_s (where $\sum_{i\in I_s}(N_i^s) = T_{abscissa}$). We repeat until the threshold is met. See the Appendix for a description of the method to calculate $APCS$ and N_i^s using the OCBA model of Chen and Lee [7]. Following is the pseudo-code for this method:

Algorithm Refine Abscissa:
Input: Δ_s

```
1.  while true:
2.      Determine $\Phi(\Delta_s)$ and calculate $APCS(\Phi(\Delta_s))$
3.      if $APCS(\Phi(\Delta_s)) \geq \tau_{abscissa}$: break
4.      else:
5.          Calculate $N_i^s$ for $i = 1, 2, \ldots, I_s$ from computational budget $T_{abscissa}$
6.          Conduct $N_i^s$ function evaluations, updating sample means and variances
7.          Update $\Phi(\Delta_s)$ and recalculate $APCS(\Phi(\Delta_s))$
8.      end if;
9.  end while;
10. return $\Phi(\Delta_s)$
```

3.2. Refining the Incumbent Solution

Though refining each abscissa generally increases the probability of correctly selecting each $\Phi(\Delta_s)$, this does not necessarily increase the probability of correctly selecting the incumbent solution $\mathbf{c}^*$. While $\mathbf{c}^*$ will be among the $\Phi(\Delta_s) \in \Theta$, we have not yet compared each $\Phi(\Delta_s)$. Refinement may be necessary to choose the correct incumbent solution with a prescribed level of probability (see Figure 3(c)). Correctly selecting the incumbent will increase the probability of correctly selecting only those hyper-rectangles satisfying (4). In practice, this can reduce the effort expended searching less-promising regions of the solution space.

Following essentially the same method as **Algorithm Refine Abscissa**, this second mechanism for improving the performance of DIRECT-S first calculates the approximate probability of correctly selecting $\mathbf{c}^*$, $APCS(\mathbf{c}^*)$, using the method in the Appendix. If this value is below $\tau_{incumbent}$, we expend $T_{incumbent}$ function evaluations, allocated as N_i^s. We repeat until the threshold is satisfied. Following is the pseudo-code for this method:

Algorithm Refine Incumbent:
Input: Θ, i.e., the set of all $\Phi(\Delta_s), s \in S$

```
1.  while true:
2.      Determine incumbent $\mathbf{c}^*$ and calculate $APCS(\mathbf{c}^*)$
3.      if $APCS(\mathbf{c}^*) \geq \tau_{incumbent}$: break
4.      else:
5.          Calculate $N_i^s$ for $i = 1, 2, \ldots, I_s$ from computational budget $T_{incumbent}$
6.          Conduct $N_i^s$ function evaluations, updating sample means and variances
7.          Update incumbent $\mathbf{c}^*$ and recalculate $APCS(\mathbf{c}^*)$
8.      end if;
9.  end while;
10. return incumbent solution $\mathbf{c}^*$
```

3.3. Filtering Based on Second Optimality Condition

Our third mechanism for improving the performance of DIRECT-S filters from Π those elements i on the convex hull that do not have a sufficiently high probability of satisfying (4), that is,

$$P\{f(\mathbf{c}_i) - \tilde{k}d_i \le f(\mathbf{c}^*) - \epsilon|f(\mathbf{c}^*)|\}. \tag{5}$$

We approximate this probability in the same manner as the probability of correctly selecting each $\Phi(\Delta_s)$ (see Appendix). If this probability is at or above the threshold value τ_{filter}, we include the hyper-rectangle in Π. We do not expend additional function evaluations to refine this probability, as we have found in practice that such effort is usually unproductive. Following is the pseudo-code for this method:

Algorithm Filter:
Input: Set of hyper-rectangles i on convex hull; incumbent solution $\mathbf{c}^*$

```
1.  for $i$ on convex hull:
2.      if $P\{f(\mathbf{c}_i) - \tilde{k}d_i \le f(\mathbf{c}^*) - \epsilon|f(\mathbf{c}^*)|\} \ge \tau_{filter}$:
3.          $\Pi \leftarrow \Pi \cup i$
4.      end if;
5.  end for;
6.  return $\Pi$, i.e., set of potentially optimal hyper-rectangles
```

3.4. DIRECT-S Algorithm

The following pseudo-code describes our overall algorithm:

Algorithm DIRECT-S:
Input: τ threshold values, ϵ value, maximum number of function evaluations *maxIterations*

```
1.  Normalize solution space and create original unit hyper-cube $i = 1$
2.  Evaluate $f(\mathbf{c}_1)$ and add original unit hyper-cube to $\Delta_1$
3.  while ($iteration \le maxIterations$):
4.      $previousIncumbent \leftarrow \{\}$
5.      while true:
6.          $\Pi \leftarrow \{\}$; $\Theta \leftarrow \{\}$
7.          for $\Delta_s \in S$:
8.              $\Phi(\Delta_s) \leftarrow$ Algorithm Refine Abscissa ($\Delta_s$)
9.              $\Theta \leftarrow \Theta \cup \Phi(\Delta_s)$
10.         end for;
11.         $\mathbf{c}^* \leftarrow$ Algorithm Refine Incumbent($\Theta$)
12.         if $\mathbf{c}^* = previousIncumbent$: break
13.         else: $previousIncumbent \leftarrow \mathbf{c}^*$
14.         end if;
15.     end while;
16.     Calculate convex hull
17.     $\Pi \leftarrow$ Algorithm Filter($i$ on convex hull; $\mathbf{c}^*$)
18.     for ($i \in \Pi$):
19.         Evaluate $f(\mathbf{c}_i \pm \delta\mathbf{e}_r)$ and divide $i$ into smaller hyper-rectangles
20.     end for;
21.     $iteration \leftarrow iteration + 1$
22. end while;
23. return $\mathbf{c}^*$ and $f(\mathbf{c}^*)$, i.e., location and objective value of best solution found
```

At each iteration, DIRECT-S enters the refinement phase (steps 5-15) to iteratively refine each $\Phi(\Delta_s)$ and the incumbent solution $\mathbf{c}^*$ until *previousIncumbent* has not changed. In step 16 it calculates the convex hull from Θ, and in step 17 it applies **Algorithm Filter** to create the set Π. In steps 18-20 it samples and divides each $i \in \Pi$ before beginning another iteration.

3.5. Convergence

As with deterministic DIRECT, DIRECT-S will always sample from that Δ_s comprising the largest hyper-rectangles, and thus will eventually converge within an arbitrary distance of any point, including the optimum. However, neither algorithm can guarantee that any particular point is the true global optimum. In addition, due to the inherent uncertainty of the objective function, our algorithm cannot guarantee that any point is a local optimum. It may be possible to devise a sampling rule that can provide almost certain convergence, following Kim and Zhang [30]. We recommend this for future research.

4. Numerical Analysis

We implement our algorithm in `Python` and use the `mpmath` library (Johansson et al. [28]) to enable storage and calculation of arbitrary-precision floating point numbers. This is useful in later phases of search when it is increasingly difficult to calculate differences between the size, sample means, and sample variances of proximate hyper-rectangles.

DIRECT-S has a number of input parameters. A full sensitivity analysis is beyond the scope of this paper; however in limited testing we find DIRECT-S is fairly robust to minor changes in these values. Unless otherwise noted, we use the following input parameters:

- Initial number of function evaluations per point: 3
- τ thresholds: 0.7
- T during each refinement: 10 + number of hyper-rectangles being refined

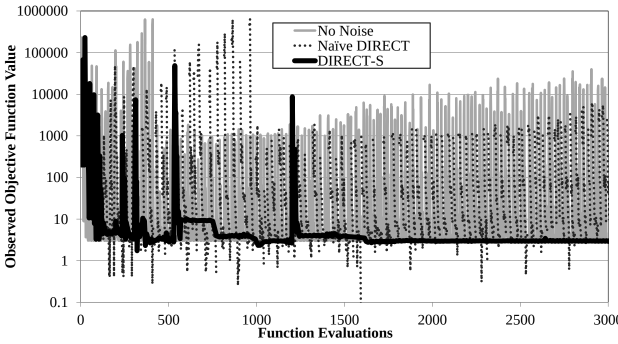
By using only three function evaluations for the initial sample of a point, we maximize the computational budget remaining for further search and the refinement process.

4.1. Visualization of Search Process

To provide insight into how DIRECT-S searches the solution space, we provide a visualization of the algorithm computing a noisy function. We consider the two-dimensional *Goldstein-Price* test function [18], adding normally-distributed noise of the form $N(\mu = 0, \sigma^2 = 10)$. The optimal solution objective value is $f(\mathbf{c}^*) = 3$, occurring at $\mathbf{c}^* = (0, -1)$. We compare DIRECT-S to a naïve implementation of deterministic DIRECT that simply samples each point three times, and to deterministic DIRECT in solving the original, noiseless function. The results are presented in Figure 4, where the vertical axis indicates the observed sample objective function value at each function evaluation. Deterministic DIRECT – exploring the noiseless function – is able to expend all function evaluations on searching the solution space and thus remains global. DIRECT-S clearly focuses on points near the global optimum, while the naïve implementation of deterministic DIRECT maintains a global search without refining any existing solutions. Naïve DIRECT has a strong tendency to get stuck in invalid local optima due its small, fixed sample size and inability to reduce the sample variance of any solution.

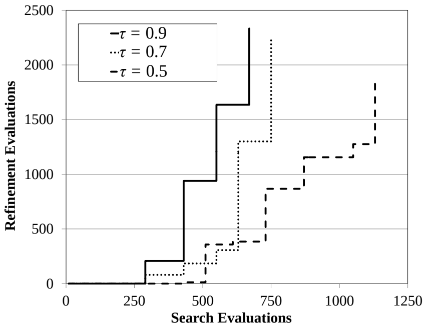
After 3,000 function evaluations, deterministic DIRECT (without noise) obtains a solution within 1.24×10^{-7} of the optimum. Naïve DIRECT becomes stuck in a false minimum, with an observed function value of -3.667 and an actual (i.e., noiseless) value of 3.1696. DIRECT-S obtains a solution objective value (2.917) within .083 of the true global minimum. Further, the solution obtained by DIRECT-S is an order of magnitude closer (i.e., in Euclidean space) to the actual global minimum solution point than that obtained by naïve DIRECT.

FIGURE 4. Observed objective function values using naïve DIRECT and DIRECT-S on deterministic and noisy Goldstein-Price test function.



Note. Without noise, deterministic DIRECT is able to expend all function evaluations sampling new points, thus its search remains global. DIRECT-S expends evaluations refining solutions occurring near the actual function minimum at $f(\mathbf{c}^*) = 3$, whereas the naïve implementation of deterministic DIRECT never refines solutions.

FIGURE 5. Cumulative function evaluations expended on search vs. refinement using noisy Goldstein-Price test function.



Note. As refinement thresholds $\tau_{abscissa}$ and $\tau_{incumbent}$ increase, an increasing proportion of the total computational budget is expended on refining solutions, i.e., reducing sample variance for previously-discovered solutions.

4.2. Search Versus Refinement

As noted earlier, a fundamental tradeoff in stochastic simulation optimization is determining the amount of computational effort expended on searching the solution space versus refining existing solutions. In DIRECT-S, this tradeoff is controlled by selection of refinement threshold values $\tau_{abscissa}$ and $\tau_{incumbent}$. To demonstrate this, we plot in Figure 5 the cumulative number of function evaluations expended on refining solutions (vertical axis) and sampling new solutions (horizontal axis) using the Goldstein-Price test function, for τ values of 0.5, 0.7, and 0.9 and initial function samples of ten function evaluations. As the thresholds τ increase, so too in general does the amount of effort expended on refinement. Note that higher levels of noise will have the same effect, requiring additional computation to reduce sample variance.

4.3. Performance Comparison

We compare the performance of DIRECT-S to the Noisy DIRECT algorithm of Deng and Ferris [12]. We use the Goldstein-Price test function with normally distributed noise at $\sigma^2 = 10$, running each algorithm for 3,000 function evaluations. Table 1 presents a comparison of the incumbent solutions of DIRECT-S and Noisy DIRECT for algorithm iterations k, where $f(x_k)$ and $\bar{f}(x_k)$ are respectively the actual underlying objective function values (i.e., no noise) and the observed sample mean, and *FN* indicates the total number of function evaluations. During the first four iterations, the algorithms obtain similar answers using the same number of function evaluations since their search methods up to this point are the same. Beyond the fourth iteration, the methods and obtained incumbents diverge. At each step, DIRECT-S obtains a more accurate solution than Noisy DIRECT, and does so in fewer total function evaluations.

We next compare DIRECT-S to Snobfit (Huyer and Neumaier [25]) and two versions of Noisy DIRECT: the original based on the normal distribution and a second based on the Student's t distribution [12]. Each algorithm is run using 3,000 total function evaluations. The process is repeated ten times; the average of the best obtained solutions are presented in Table 2. Column FN_i indicates the initial number of function evaluations per point, *Obj error* is the absolute difference of the actual underlying objective function value and the global optimum, and *Distance* is the Euclidean distance from the sampled point and the

Table 1. Comparison of incumbent solutions to Goldstein-Price test function at iteration k.

	Noisy DIRECT			DIRECT-S		
k	$f(x_k)$	$\bar{f}(x_k)$	FN	$f(x_k)$	$\bar{f}(x_k)$	FN
1	200.54	201.03	15	200.55	197.9	15
2	200.54	201.03	21	200.55	197.9	21
3	14.92	12.68	39	8.92	10.67	39
4	14.92	12.68	63	8.92	10.67	63
5	3.64	7.16	81	3.65	3.27	69
6	3.64	7.16	111	3.65	3.27	87
7	5.84	-0.74	298	3.06	3.95	105
8	4.55	2.13	380	3.06	3.34	234
9	3.55	3.89	1270	3.01	2.82	306
10	3.55	3.72	3010	3.01	2.66	529
11				3.01	2.92	1201
12				3.01	2.92	3021

Note. At iteration $k = 5$ and beyond, DIRECT-S samples different points than Noisy DIRECT and is able to consistently obtain a better incumbent solution.

Table 2. Comparison of best obtained solutions to Goldstein-Price test function after 3,000 total function evaluations.

Algorithm	FN$_i$	Obj error	Distance
DIRECT-S	3	0.0569	0.0125
Noisy DIRECT	Auto	0.3787	0.0288
Noisy DIRECT(t)	Auto	0.3445	0.0283
Naïve DIRECT	1	2.4570	0.0694
	5	1.5045	0.0601
	10	0.6119	0.0432
	50	0.4073	0.0296
	100	0.6474	0.0370
Snobfit	1	2.3231	0.0688
	5	2.0332	0.0802
	10	0.9761	0.0536
	50	11.683	0.1805
	100	44.456	0.5695

Note. DIRECT-S obtains a better objective value than every method except Naïve DIRECT with 50 initial function evaluations, and also obtains a point closer to the true global optimum.

Table 3. Comparison of obtained solutions for several test functions.

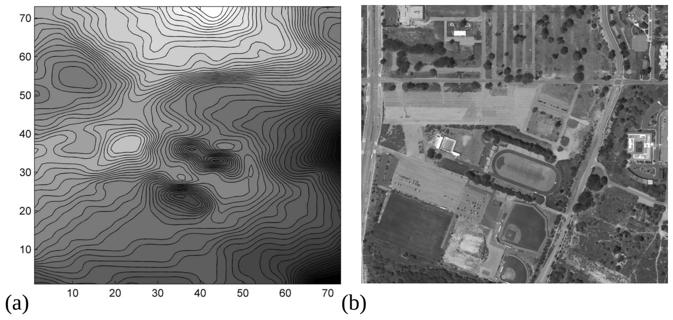
	Algorithm A-F				DIRECT-S			
Problem	k	FN	Obj error	Distance	k	FN	Obj error	Distance
Rosenbrock	20	682	1.9	2.1	11	589	0.102	0.831
Camel	21	710	0.038	0.13	8	509	0.036	0.137
Powell	29	1932	0.11	0.4	6	683	0.106	0.123

Note. DIRECT-S finds comparable or better solutions than the method of Anderson and Ferris [2] in considerably fewer function evaluations.

true solution. DIRECT-S obtains a more accurate solution than any method except Naïve DIRECT with 50 initial function evaluations, though DIRECT-S is closer to the actual true solution. Both Naïve DIRECT and Snobfit require more initial function evaluations than DIRECT-S in order to obtain good solutions, a drawback if the simulation is particularly expensive.

Next, we compare DIRECT-S against the method proposed by Anderson and Ferris [2], referred to as *Algorithm A-F*. Like DIRECT-S, their method is useful for black-box optimization and requires only that the objective function be Lipschitz continuous; they prove local convergence of their algorithm in the presence of normally-distributed noise. A comparison of their method and DIRECT-S is presented in Table 3 for three test functions (each with added normally-distributed noise at $\sigma = 0.1$): the *Rosenbrock* function [36], the six-hump camelback function *Camel*, and the *Powell* function [41]. In each case, DIRECT-S is able to obtain a solution closer to the global minimum in considerably fewer function evaluations.

FIGURE 6. Wireless mesh network operating area at Fort Ord, California.



Note. An elevation contour map (a) and Google Maps [35] image (b) of the operating area.

5. A Computational Example: Wireless Mesh Network Design

We provide an application of DIRECT-S to demonstrate its ability to solve a realistic simulation optimization problem. *Wireless mesh networks* (*WMNs*) are interconnected systems of wireless *access points* (*APs*) frequently used to provide contiguous WiFi service across a broad area, e.g., a campus or airport. Each AP provides wireless connectivity to nearby client devices, including laptops and smart phones. Each AP also wirelessly interconnects to other APs, creating a backhaul network used to exchange communications traffic. The problem of WMN design (i.e., where to physically position APs) is difficult due to numerous factors, including the nonlinear effects of radio propagation and the unpredictability of client traffic demands.

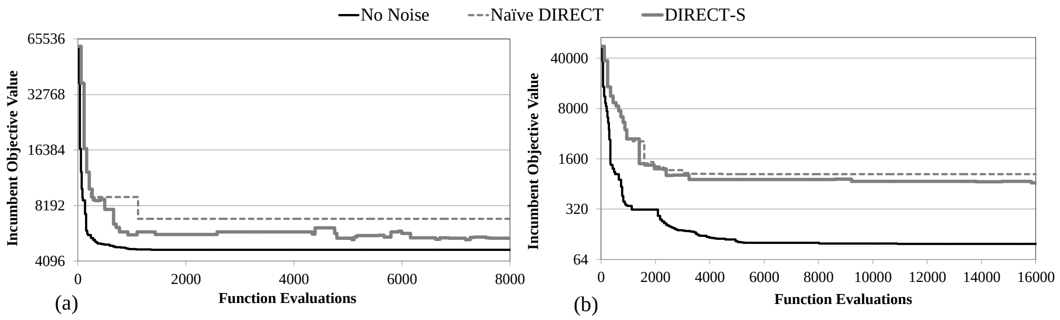
We quantify the value of WMN topologies using the methodology and software of Nicholas and Alderson [39, 40]. In their *simultaneous routing, resource allocation, and coverage* (*SRRA+C*) formulation, APs must be placed to simultaneously maximize client coverage area and delivered network service. The software uses the *Terrain Integrated Rough Earth Model* (*TIREM*) [1] to accurately predict radio propagation effects through the atmosphere and over rough terrain, and has been used to design actual WMNs [38, 39]. We modify their formulation to consider stochastic network traffic, where each region receiving client service has normally-distributed traffic demand. We then connect DIRECT-S to the SRRA+C software to enable the algorithm to iteratively sample the SRRA+C solution space in pursuit of optimal topologies.

We use this framework to design a WMN at Fort Ord, California using Cisco Aironet 1550 WMN APs [26]. We consider a rectangular operating area of 116 acres, consisting of gently rolling hills, a large parking lot, a stadium, and several buildings (see Figure 6). We consider networks for five and ten total APs; the location of each AP is described using a northing and easting, so there are respectively 10 and 20 dimensions in these problems.

We solve using DIRECT-S with 8,000 and 16,000 total function evaluations for the five AP and ten AP problems. We compare our results to naïve DIRECT and the noiseless version of each problem in Figure 7. In both problems, DIRECT-S is able to find a more desirable WMN topology than naïve DIRECT (note SRRA+C is a minimization problem). In the five AP problem (Figure 7(a)), DIRECT-S is able to find a solution with an objective value within 21% of the global optimum after 1,411 function evaluations; naïve DIRECT becomes stuck in a local minimum at 1,113 function evaluations with a solution objective value 47% greater than the global optimum. The ten AP problem (Figure 7(b)) is decidedly more difficult to solve due to the greater number of dimensions. After nearly 16,000 function evaluations, DIRECT-S finds a solution that is 12.3% closer to the global optimum than the best solution found by naïve DIRECT, which falls into a local minimum after 4,503 function evaluations.

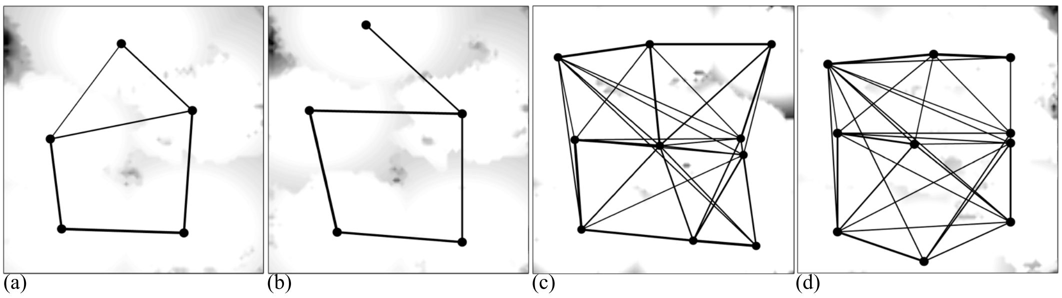
Next we compare the topologies found using DIRECT on the deterministic version of the problem (Figures 8(a) and (c)) and DIRECT-S on the stochastic version (Figures 8(b) and

FIGURE 7. Comparison of optimization methods for designing wireless mesh networks.



Note. DIRECT-S finds a better solution than naïve DIRECT for wireless mesh networks of five APs (a) and ten APs (b).

FIGURE 8. Relative AP locations with predicted client coverage and network connectivity.



Note. The five AP network without (a) and with noise (b), and the ten AP network without (c) and with noise (d). Shaded areas indicate inadequate client coverage; the width of each line indicates relative network throughput between each AP.

(d)). Shaded areas indicate regions receiving less than adequate client coverage; the width of each line indicates the relative predicted network throughput between each AP. DIRECT-S finds topologies that are very similar to those found with the noiseless function.

6. Conclusions and Areas for Future Research

We demonstrate that the DIRECT-S algorithm is capable of outperforming the other known stochastic implementation of the DIRECT algorithm. While DIRECT-S is fairly robust to minor changes in input parameters, we recommend a thorough and systematic exploration of the parameter space to determine if there are general trends that could inform the choice of parameter values, or perhaps determine these values dynamically. For instance, it seems sensible to gradually increase the refinement thresholds τ as the algorithm progresses toward localized search.

One disadvantage of DIRECT-S is that it is practically applicable only to low-dimensional problems because it suffers from the *curse of dimensionality* [6, 20]. Some research has been done to overcome this via parallelization of the algorithm (see, e.g., Gablonsky [17], Griffin and Kolda [19], Watson and Baker [50]), and by breaking the search space into smaller regions (see, e.g., Tavassoli et al. [46]). This problem is exacerbated for stochastic functions, as plural sampling of each point is required to account for noise.

While the algorithm is guaranteed to eventually sample within an arbitrary distance of the optimal solution, it cannot provide a certificate of optimality for any particular solution. Future research could devise a rule for selecting sample sizes that provide almost certain convergence to a local optimum (see, e.g., Kim and Zhang [30]). An apparently achievable step toward this proof would be the derivation of a more precise method of calculating the probability of correctly selecting the convex hull.

It would be interesting to compare DIRECT-S to other black-box optimization methods, including *compass search* (see, e.g., Davidon [10], Kolda et al. [32], and Torczon [47]), *mesh adaptive direct search* (*MADS*) (Audet and Dennis [3]), the *Nelder-Mead simplex method* (Nelder and Mead [37], Lagarias et al. [33], Barzinpour et al. [5], and Wang et al. [49]), the methods of Lucidi and Sciandrone [34], the *nested partitions algorithm* (Shi and Ólafsson [44, 45] and Shi and Chen [43]), and *partition-based random search* (Chen et al. [9]).

7. Acknowledgments

The author is indebted to Professor C.H. Chen for providing instruction and insight into his OCBA methodology. The author also thanks Professor K. Hoffman and several colleagues and reviewers for constructive comments on this and earlier versions of this paper.

Appendix

A. Calculating the Probability of Correct Selection

We calculate the probability of correctly selecting each $\Phi(\Delta_s)$ following the Bayesian model of Chen and Lee [7]. Let $f(\mathbf{c}_i^s)$ be the unknown true mean of ith hyper-rectangle within Δ_s, for $i = 1, 2, \ldots, I_s$. We assume $f(\cdot)$ has a conjugate normal prior distribution, and the observed sample performance estimates $F(\cdot, \omega)$ are normally distributed and independent. The posterior distribution of $f(\mathbf{c}_i^s)$ is

$$\tilde{f}(\mathbf{c}_i^s) \sim N\left(\bar{f}(\mathbf{c}_i^s), \frac{(\sigma_i^s)^2}{N_i^s}\right) \tag{6}$$

where $\bar{f}(\mathbf{c}_i^s)$ is the sample mean of $f(\mathbf{c}_i^s)$, $(\sigma_i^s)^2$ is the variance (estimated by sample variance in practice), and N_i^s is the number of function evaluations of the i^{th} hyper-rectangle. Let b be the index of the $\Phi(\Delta_s)$ hyper-rectangle, that is

$$b = \arg\min_{i \in \Delta_s} \bar{f}(\mathbf{c}_i^s). \tag{7}$$

The probability of correctly selecting each $\Phi(\Delta_s)$ is:

$$P\{\text{CS } \Phi(\Delta_s)\} = P\{\tilde{f}(\mathbf{c}_b^s) < \tilde{f}(\mathbf{c}_i^s), i \neq b\}. \tag{8}$$

As Chen and Lee [7] observe, this probability can be estimated for more than two designs using Monte Carlo simulation, but this can be time-consuming. We follow their method and calculate the *approximate probability of correct selection* (*APCS*) using the following product:

$$APCS\{\Phi(\Delta_s)\} \equiv \prod_{1 \le i \le I_S,\, i \neq b} P\{\tilde{f}(\mathbf{c}_b^s) < \tilde{f}(\mathbf{c}_i^s)\}. \tag{9}$$

It can be shown this provides a lower bound on $P\{\text{CS } \Phi(\Delta_s)\}$ [8]. A similar approach is used to calculate the $APCS\{\mathbf{c}^*\}$ and to approximate (5).

B. Allocating the Computational Budget

We wish to allocate the available computational budget T among the given hyper-rectangles $i \in I_s$ in order to maximize the probability of correctly selecting the best (i.e., lowest) function value $f(\mathbf{c}_i^s)$. Following Chen and Lee [7], the $APCS$ $\Phi(\Delta_s)$ for a given Δ_s can be asymptotically maximized according to the following OCBA rules for allocating the available function evaluations:

$$\frac{N_i^s}{N_j^s} = \left(\frac{\sigma_i^s/\delta_{b,i}^s}{\sigma_j^s/\delta_{b,j}^s}\right)^2, \quad i, j \in \{1, 2, \ldots, I_s\},\ i \neq j \neq b, \tag{10}$$

$$N_b^s = \sigma_b^s \sqrt{\sum_{1 \le i \le I_S,\, i \neq b} \frac{(N_i^s)^2}{(\sigma_i^s)^2}} \tag{11}$$

where N_i^s is the relative number of function evaluations allocated to hyper-rectangle i and $\delta_{b,i}^s$ is the difference in sample mean between hyper-rectangles b and i. The computational budget T is allocated according to the ratio $\{N_b^s : N_2^s : N_3^s : \cdots : N_{I_s}^s\}$. Upon calculating this budget, we conduct the given number of function evaluations and update the sample running means and running variances using the method of Welford [51] and Knuth [31].

References

[1] Alion Science and Technology Corporation. TIREM Details. World Wide Web, `http://www.alionscience.com/en/Technologies/Wireless-Spectrum-Management/TIREM/`.

[2] E.J. Anderson and M.C. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal on Optimization*, 11(3):837–857, 2001.

[3] C. Audet and J.E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[4] F. Esmaeilzadeh Azar and M. Perrier. Extension of the global optimization using multi-unit extremum seeking control for noisy scalar systems. In *European Control Conference (ECC)*, pages 1236–1241, 2013.

[5] F. Barzinpour, R. Noorossana, S.T.A. Niaki, and M.J. Ershadi. A hybrid Nelder–Mead simplex and PSO approach on economic and economic-statistical designs of MEWMA control charts. *The International Journal of Advanced Manufacturing Technology*, 65(9-12):1339–1348, 2013.

[6] R.E. Bellman. *Adaptive Control Processes: A Guided Tour*, volume 4. Princeton University Press, 1961.

[7] C.H. Chen and L.H. Lee. *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation*. World Scientific, 2011.

[8] C.H. Chen, J. Lin, E. Yücesan, and S.E. Chick. Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 10:251–270, 2000.

[9] W. Chen, S. Gao, C.H. Chen, and L. Shi. An optimal sample allocation strategy for partition-based random search. *IEEE Transactions on Automation Science and Engineering*, 11(1):177–186, 2014.

[10] W.C. Davidon. Variable metric method for minimization. Technical Report 5990, Argonne National Laboratory, Lemont, IL, 1959.

[11] G. Deng. *Simulation-based optimization*. PhD thesis, University of Wisconsin–Madison, 2007.

[12] G. Deng and M.C. Ferris. Extension of the DIRECT optimization algorithm for noisy functions. In *Proceedings of the 2007 Winter Simulation Conference*, pages 497–504, 2007.

[13] D. di Serafino, G. Liuzzi, V. Piccialli, F. Riccio, and G. Toraldo. A modified dividing rectangles algorithm for a problem in astrophysics. *Journal of Optimization Theory and Applications*, 151(1):175–190, 2011.

[14] D.E. Finkel. DIRECT optimization algorithm user guide. *Center for Research in Scientific Computation, North Carolina State University*, 2003.

[15] D.E. Finkel and C.T. Kelley. Convergence analysis of the DIRECT algorithm. *Optimization Online*, pages 1–10, 2004.

[16] M.C. Fu, C.H. Chen, and L. Shi. Some topics for simulation optimization. In *Proceedings of the 2008 Winter Simulation Conference*, pages 27–38, 2008.

[17] J.M Gablonsky. *Modifications of the DIRECT algorithm*. PhD thesis, 2001.

[18] A.A. Goldstein and J.F. Price. On descent from local minima. *Mathematics of Computation*, 25(115), 1971.

[19] J.D. Griffin and T.G. Kolda. Asynchronous parallel hybrid optimization combining DIRECT and GSS. *Optimization Methods & Software*, 25(5):797–817, 2010.

[20] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer Series in Statistics, 2009.

[21] J. He, A. Verstak, L. Watson, C. Stinson, N. Ramakrishnan, C. Shaffer, T. Rappaport, C. Anderson, K. Bae, J. Jiang, and W. Tranter. Globally optimal transmitter placement for indoor wireless mesh networks. *IEEE Transactions on Wireless Communications*, 3(6):1906–1911, 2004.

[22] R. He and P.A. Narayana. Global optimization of mutual information: application to three-dimensional retrospective registration of magnetic resonance images. *Computerized Medical Imaging and Graphics*, 26(4):277–292, 2002.

[23] T. Hemker and C. Werner. DIRECT using local search on surrogates. *Pacific Journal of Optimization*, 7(3):443–466, 2011.

[24] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, 1996.

[25] W. Huyer and A. Neumaier. Snobfit–stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)*, 35(2):9, 2008.

[26] Cisco Systems Inc. Cisco 1550 Series Outdoor Access Point Data Sheet. World Wide Web, `http://www.cisco.com/`.

[27] H. Jeffreys and B. Jeffreys. *Methods of Mathematical Physics*. Cambridge University Press, 1999.

[28] F. Johansson et al. mpmath: a Python library for arbitrary-precision floating point arithmetic. World Wide Web, `http://mpmath.org/`, 2013.

[29] 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.

[30] S. Kim and D. Zhang. Convergence properties of direct search methods for stochastic optimization. In *Proceedings of the 2010 Winter Simulation Conference*, pages 1003–1011, 2010.

[31] D.E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, 2014.

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

[33] J.C. Lagarias, J.A. Reeds, M.H. Wright, and P.E. Wright. Convergence properties of the Nelder–Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[34] S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Computational Optimization and Applications*, 21(2):119–142, 2002.

[35] Google Maps. World Wide Web, `http://maps.google.com/`, 2013.

[36] J.J. Ḿore, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7:17–41, 1981.

[37] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[38] P.J. Nicholas. Optimal transmitter placement in wireless mesh networks. M.S., Naval Postgraduate School, Monterey, CA, 2009.

[39] P.J. Nicholas and D.L. Alderson. Fast, effective transmitter placement in wireless mesh networks. *Military Operations Research*, 17(4):69–84, 2012.

[40] P.J. Nicholas and D.L. Alderson. Method for optimal transmitter placement in wireless mesh networks. U.S. patent 8,654,672, February 18 2014.

[41] M.J.D. Powell. An iterative method for finding stationary values of a function of several variables. *The Computer Journal*, 5:147–151, 1962.

[42] N. Saber and J.M. Shaw. Rapid and robust phase behaviour stability analysis using global optimization. *Fluid Phase Equilibria*, 264(1):137–146, 2008.

[43] L. Shi and C.H. Chen. A new algorithm for stochastic discrete resource allocation optimization. *Discrete Event Dynamic Systems*, 10(3):271–294, 2000.

[44] L. Shi and S. Ólafsson. Nested partitions method for global optimization. *Operations Research*, 48(3):390–407, 2000.

[45] L. Shi and S. Ólafsson. Nested partitions method for stochastic optimization. *Methodology and Computing in Applied Probability*, 2(3):271–291, 2000.

[46] A. Tavassoli, K. Haji Hajikolaei, S. Sadeqi, G.G. Wang, and E. Kjeang. Modification of DIRECT for high-dimensional design problems. *Engineering Optimization*, 46(6):810–823, 2014.

[47] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.

[48] M.P. Wachowiak and T.M. Peters. High-performance medical image registration using new optimization techniques. *IEEE Transactions on Information Technology in Biomedicine*, 10(2): 344–353, 2006.

[49] L. Wang, Y. Xu, and L. Li. Parameter identification of chaotic systems by hybrid Nelder–Mead simplex search and differential evolution algorithm. *Expert Systems with Applications*, 38(4):3238–3245, 2011.

[50] L.T. Watson and C.A. Baker. A fully-distributed parallel global search algorithm. *Engineering Computations*, 18(1/2):155–169, 2001.

[51] B.P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[52] H. Zhu and D.B. Bogy. DIRECT algorithm and its application to slider air-bearing surface optimization. *IEEE Transactions on Magnetics*, 38(5):2168–2170, 2002.