On the Natural Extension of the DIRECT Global Optimization Algorithm to Handle Multiple Objectives, Nonlinear Constraints, and Missing Data

Donald R. Jones
Department of Aerospace Engineering
University of Michigan
Ann Arbor, Michigan

June 6, 2022

Abstract

Recently, Lovison and Miettinen [1] extended the DIRECT global optimization algorithm to handle multiple objectives, introducing a new algorithm called "*multi*DIRECT." Unlike previous multiobjective extensions, Lovison and Miettinen remained true to DIRECT's original Lipschitzian motivation and preserved its much-admired conceptual simplicity. However, while the *ideas* of the new algorithm were simple and intuitive, the *algorithmic implementation* was complex and difficult to follow. In this paper, we introduce a simplified version called "simDIRECT" (for "**si**mplified **m**ultiobjective **DIRECT**") that follows the same basic idea, but is much easier to understand and implement. In addition, we show how simDIRECT can be adapted to handle nonlinear equality and inequality constraints, and also to solve problems in which the objective and/or constraint functions sometimes fail to compute. Initial experience with simDIRECT on test problems suggests that the method performs similar to, or better than, multiobjective evolutionary algorithms when solving problems with small numbers of variables (up to 12), small number of objectives (up to 3), and a limited number of runs (up to 600).

1 Introduction

Today, perhaps the most popular algorithms for black-box, multiobjective optimization are Multi-Objective Evolutionary Algorithms (MOEAs) [2]. Evolutionary algorithms are a natural candidate for multiobjective optimization because their population-based structure allows them to evolve a set of points towards the Pareto front in a single optimization run. In effect, MOEAs search for multiple points on the Pareto front in parallel, in contrast to scalarization methods that solve a series of single-objective optimization problems, each of which returns a single point on the Pareto front. While often effective, MOEAs are stochastic algorithms with many tunable parameters, so they may require multiple runs, off-line parameter tuning, or the development of schemes that adapt parameters during the optimization itself [3, 4].

For low-dimensional problems (say, up to 12 variables and 3 objectives), the DIRECT global optimization algorithm provides an attractive deterministic alternative to MOEAs. Like an MOEA, DIRECT evolves a population of points which can be made to converge to the Pareto front. But unlike MOEAs, DIRECT is deterministic, so multiple runs are not necessary; moreover, its one tunable parameter (the desired accuracy) is easy to specify and does not require tuning.

Until very recently, all proposed extensions of DIRECT to handle multiple objectives were weak in some regard. Some were slow in converging to the Pareto front, while others converged faster but had no good way to accommodate nonlinear side constraints. And none of the extensions built upon the original Lipschitzian foundation of DIRECT.

In 2021, however, Lovison and Miettinen [1] introduced a fresh extension of DIRECT to multiple objectives, called *multi*DIRECT, that was true to the original Lipschitzian motivation. In the Lovison-Miettinen approach, one selects regions of the space for further search based on whether or not they satisfy set of conditions that are the natural extension of the conditions used in the single-objective case. With just one objective, the regions meeting these conditions can be visualized as the lower-right convex hull of a set of points in a certain two-dimensional diagram (more on this later), and efficient algorithms are available to find the convex hull. With $m \geq 2$ objectives, the corresponding diagram has $m+1$ dimensions and the required geometric reasoning to determine if a region meets the conditions is much more complex. Nevertheless, Lovison and Miettinen were able to implement their method

and showed good results. However, given the computational complexity of the method, adding side constraints would be difficult.

Inspired by *multi*DIRECT's conceptual appeal, but overwhelmed by its computational complexity, I looked for a way to simplify the method without losing the connection to DIRECT's Lipschitzian beginnings. By borrowing a trick from how DIRECT was previously extended to handle constraints [5], I was able to produce a simplification of *multi*DIRECT in which the calculations in $m+1$ dimensions where reduced to calculations in just one dimension. Moreover, this simplified version could easily accommodate nonlinear side constraints; in fact, on some single-objective constrained problems, it often outperforms my earlier extension of DIRECT to handle nonlinear constraints [5]. Equally exciting, I found a way to handle problems in which the objective and constraint functions sometimes fail to compute – sometimes referred to as "hidden constraints" [6, 7, 8]. I call this new method "simDIRECT," short for **si**mplified **m**ultiobjective **DIRECT**.

In what follows, I will begin by briefly recalling how DIRECT works, focussing on the key Lipschitzian motivation that I strive to maintain in simDIRECT. I then review three previous multiobjective extensions of DIRECT published before Lovison and Miettinen, identifying their strengths and weaknesses. All this sets the stage for describing *multi*DIRECT and simDIRECT. Numerical examples are given throughout to illustrate and compare the various methods and to explore how well simDIRECT scales with the number of variables and objectives.

2 Single-objective DIRECT

The original DIRECT algorithm minimized a black-box objective function subject to lower and upper bounds on the variables [9]; that is, it solved the problem:

$$\begin{aligned} \underset{x_1,\ldots,x_d}{\text{minimize}} \quad & f(x_1,\ldots,x_d) \\ \text{subject to} \quad & x_k^L \le x_k \le x_k^U \quad k = 1,\ldots,d. \end{aligned}$$

where d is the number of variables. Several years later, the same authors published a revision of DIRECT that added the capability to handle nonlinear inequality constraints and integer variables, as well as a few other changes to speed up convergence [5]. Given that the revised version is arguably better, we will describe that version and use it in the numerical work reported later.

Because DIRECT assumes lower and upper bounds we can, without loss of generality, normalize the design variables to $[0, 1]$ so that the search space becomes the unit hypercube. DIRECT works by partitioning the unit hypercube into subrectangles with the property that the objective function has been evaluated at each rectangle's center point. In each iteration, certain "potentially optimal rectangles" are selected for further search; these rectangles are then subdivided and the function is evaluated at the center points of the newly-formed subrectangles.

Figure 1 illustrates this process, showing the first three iterations of DIRECT on a two-variable test function. At first, there is only one rectangle (the entire unit hypercube), so we select it. This rectangle is then subdivided into thirds, and the center points of the newly formed rectangles (red dots) are evaluated. In iteration 2, just one rectangle is selected, subdivided, and sampled. In iteration 3, two rectangles are selected, and then subdivided and sampled. The process continues until we reach a specified limit on the number of iterations or function evaluations.

Clearly, the key elements of DIRECT are the method for selecting potentially optimal rectangles and the method for subdividing these rectangles.

In the example shown in Figure 1, we adopt the simplest subdivision method: select one dimension and divide the rectangle into thirds along this dimension. The original center point becomes the center of the middle third, so we only need to evaluate the objective function two new points (the centers of the newly created "left" and "right" thirds, shown as red dots).

To avoid producing skinny rectangles with high aspect ratio, we always select one of a rectangle's longer sides for trisection. For example, during iteration 2 in Figure 1, the selected rectangle had a short side in the first dimension and a long side in the second dimension. We therefore chose to trisect it along the longer second dimension. By always trisecting along a long side, the aspect ratio will never be more than 3:1.

Of course, sometimes there will be more than one long side, and we will have to decide which of these to use for trisection. We adopt the following procedure to make this decision. Let $t(\ell)$ denote the number of times we have trisected along dimension ℓ up to the current iteration. Before the search begins, we initialize $t(\ell) = 0$ for $\ell = 1, \ldots, d$. When we go to subdivide a rectangle, we first identify the set $\mathcal{L}$ of dimensions with the longest side length. We then trisect along the dimension $\ell^* \in \mathcal{L}$ that has the lowest $t(\ell)$ value and, if there is a tie, we pick the one with the lowest value of ℓ. After we trisect along the selected dimension ℓ^*, we record this by incrementing

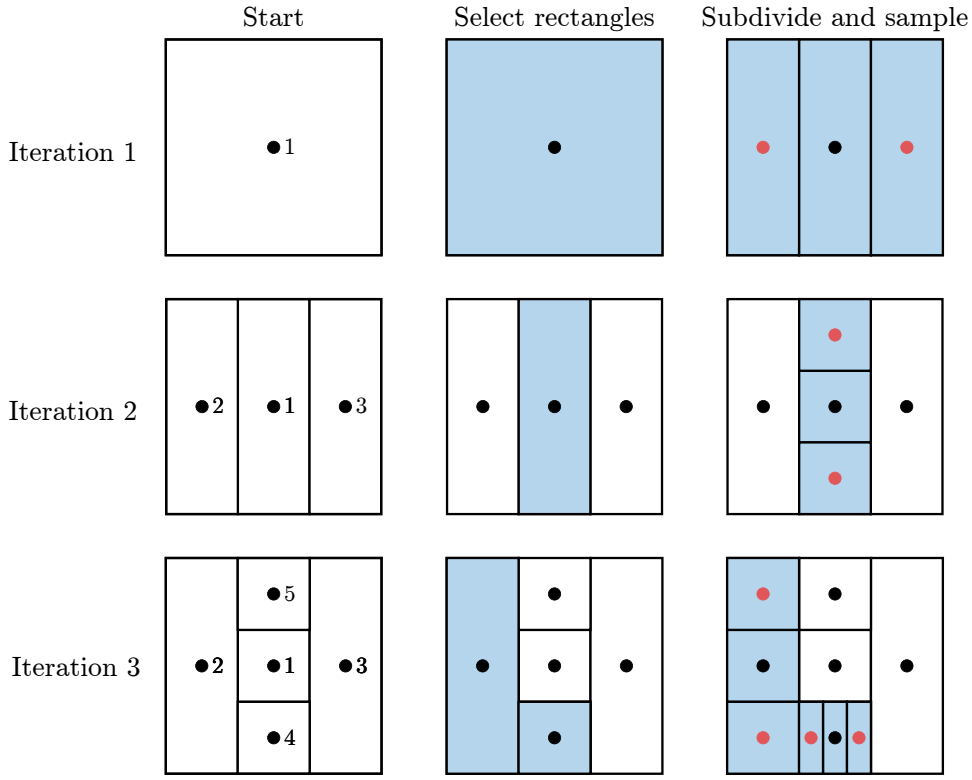

Figure 1: First three iterations of the DIRECT algorithm for a two-variable test function. The numbers next to the dots in the "Start" column indicate the number of the function evaluation.

$t(\ell^*)$. In this way we avoid favoring one dimension over another in trisection.

Let us now turn our attention to how DIRECT selects the "potentially optimal rectangles." As we mentioned earlier, DIRECT was motivated by Lipschitzian optimization so, just for the moment, let us assume that we knew a Lipschitz constant for the objective function, that is, we knew a constant K such that, for any two points $\mathbf{x}$ and $\mathbf{x}'$ we have

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq K\|\mathbf{x} - \mathbf{x}'\|. \tag{1}$$

In equation (1), the norm on the right hand side can be the standard Euclidean 2-norm or a non-Euclidean norm. The original algorithm and most extensions of it use the 2-norm, but the *multi*DIRECT algorithm described later uses the infinity norm: $\|(x_1, x_2, \ldots, x_d)\|_\infty = \max\{|x_1|, ..., |x_d|\}$. Intu-

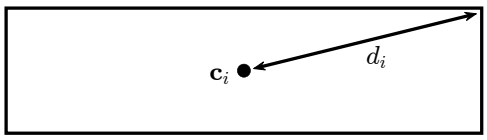

Figure 2: The center-vertex distance d_i of rectangle i.

itively, the Lipschitz constant is an upper bound on the rate of change (slope) of the function.

Now let $\mathbf{c}_i$ be the center of rectangle i, and let d_i be the Euclidean distance between $\mathbf{c}_i$ and the vertices (see Figure 2). Given that the function has a value $f(\mathbf{c}_i)$ at the center of rectangle i, and that the maximum distance between $\mathbf{c}_i$ and any point in rectangle i is d_i, it follows that a valid lower bound for f over rectangle i is:

$$\text{lower bound for } f \text{ over rectangle } i = f(\mathbf{c}_i) - Kd_i.$$

In standard Lipschitzian optimization, in each iteration we would compute such a lower bound for every rectangle and select the rectangle with the lowest lower bound for subdivision and further sampling. If K is a valid Lipschitz constant (that is, equation (1) holds for all $\mathbf{x}$ and $\mathbf{x}'$), then one can prove that, for any small positive number $\epsilon > 0$, the algorithm will find a solution within ϵ of the optimum in a finite number of iterations.

To understand how DIRECT selects rectangles, it will be helpful to visualize Lipschitzian selection with the help of the diagram in Figure 3. In this diagram, each rectangle is represented as a dot with horizontal coordinate equal to the rectangle's size (center-vertex distance d) and vertical coordinate equal to the function value (f) at the rectangle's center; thus, we call this the "f-d diagram." The gray dashed line in the figure shows the set of points where the rectangle lower bound, $f - Kd$, is equal to some constant (the constant is the value of the vertical intercept). The rectangle with the *lowest* lower bound can be found by placing this dashed line below all the dots and then sliding it up until it first touches a dot. We illustrate this sliding motion in the figure, where rectangle i turns out to be the one with the lowest lower bound.

In practice, however, one usually doesn't know a valid Lipschitz constant. DIRECT overcomes this lack of knowledge by using all possible Lipschitz constants $K > 0$. More precisely, in each iteration, DIRECT selects *any* rectangle i that has lowest Lipschitzian lower bound for *some* Lipschitz constant

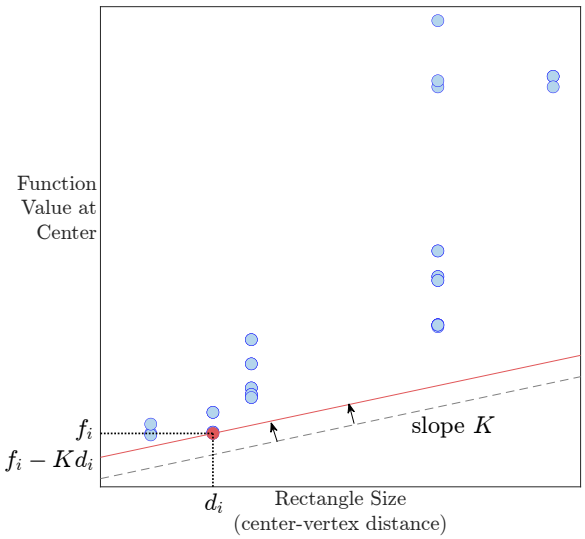

Figure 3: Standard Lipschitzian selection.

$K > 0$, subject to the constraint that the resulting lower bound be non-trivially better than the current best solution $f_{\min}$. We call the rectangles that meet these criteria "potentially optimal rectangles," defined formally as follows:

Definition 1. Potentially optimal hyperrectangle
Suppose we have a partition of the unit hypercube into n hyperrectangles and let $\epsilon > 0$ be a small positive constant. A hyperrectangle i is said to be *potentially optimal* if there exists some $K > 0$ such that

$$f(\mathbf{c}_i) - Kd_i \leq f(\mathbf{c}_j) - Kd_j, \quad \text{for all } j = 1, \ldots, n. \tag{2}$$

$$f(\mathbf{c}_i) - Kd_i \leq f_{\min} - \epsilon. \tag{3}$$

A reasonable value for ϵ would be the desired absolute accuracy in the solution, usually some small number like $\epsilon = 10^{-4}$. □

How can we identify the set of potentially optimal rectangles? Well, think back to our f-d diagram in Figure 3 and imagine sliding up a line with slope $K > 0$ until it first touches a dot, but this time do the sliding with many different values of the slope K. Clearly, the only dots that can be touched lie on the lower-right convex hull of the set of dots. Moreover, when the line first touches a dot for some rectangle i, the vertical intercept, which is

$f(\mathbf{c}_i) - Kd_i$, must be less than or equal to $f_{\min} - \epsilon$ due to equation (3). As shown in Figure 4, it follows that the set of potentially optimal rectangles correspond to those dots that lie on the lower-right convex hull of all the dots augmented with the point $(0,\ f_{\min} - \epsilon)$. In the figure, the potentially optimal rectangles are highlighted in red.

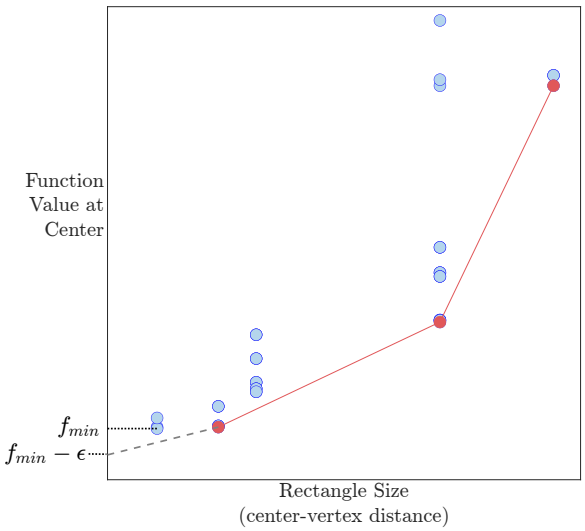

Figure 4: The selection of rectangles in DIRECT. The set of potentially optimal rectangles lie on the lower-right convex hull of the dots augmented with the point $(0,\ f_{\min} - \epsilon)$ and are highlighted in red.

There is a subtle issue that arises when implementing rectangle selection as shown in Figure 4. In particular, what looks like a single red dot in the figure (a selected rectangle) may actually be several *overlapping* dots. This could happen if several rectangles share the same size (d) and function value (f) and so overlap in the figure. In a previous article [10], I concluded that it was generally *not* good to select all the tied rectangles, but instead break the tie arbitrarily and pick just *one* (say, the lowest indexed rectangle). Intuitively, if we are lucky and select the best of the tied rectangles, we converge faster. On the other hand, if we don't select the best of the tied rectangles, nothing is lost, as we can still select it in one of the following iterations. So in single-objective optimization, breaking the ties arbitrarily often reduces the number of function evaluations needed to find a good solution.

In multiobjective optimization, it turns out that we can do something

similar to "breaking ties arbitrarily." However, in the multiobjective case, I have found that selecting *all* the tied rectangles tends to produce a better, more uniform coverage of the Pareto front, whereas breaking ties arbitrarily leads to an undesirable uneven coverage. For this reason, in what follows we will always do the equivalent of selecting all rectangles that are tied for being potentially optimal.

An interesting variation on DIRECT's selection method, suggested by Mockus [11], is to select all rectangles that are nondominated, or "Pareto optimal," with respect to the centerpoint function value (lower is better) and rectangle size (higher is better). Figure 5 shows the dots that would be selected by the Pareto method; these dots will always include the potentially optimal rectangles and sometimes a few more (in this example, two more). The downside of the Pareto method is that it lacks any mechanism such as equation (3) that prevents it from selecting very small rectangles for which we can expect only minuscule improvement in the objective. As a result, the Pareto method can sometimes waste precious function evaluations pursuing very small improvements around a suboptimal local minima. We mention Pareto selection here because, later in the paper, we discuss a multiobjective extension of DIRECT that is based on this approach.

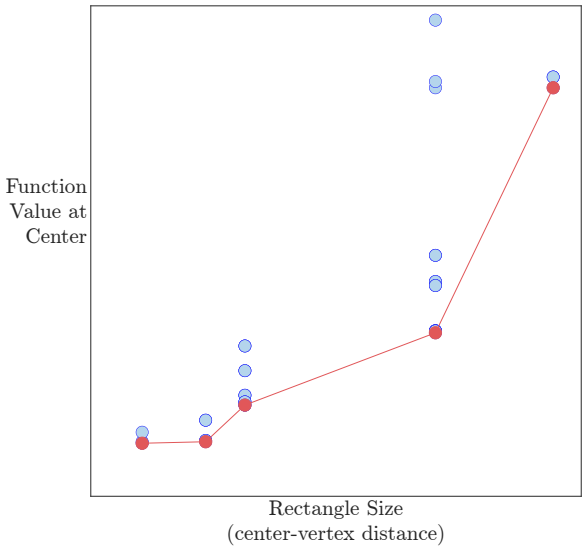

Figure 5: In Pareto selection, we select all those rectangles (red dots in the figure) that are nondominated with respect to function value at the center and rectangle size.

The effectiveness of DIRECT is not simply due to its finding a way around the problem of not knowing a valid Lipschitz constant. Instead, what makes DIRECT effective is the use of *several* Lipschitz constants in each iteration. The large values of K select big rectangles that are good for global exploration, while small values of K select smaller rectangles good for local search. In this way, once the global part of DIRECT's search finds the basin of the optimum, it is immediately exploited by the local part.

In Figure 6 we show two examples of the DIRECT algorithm solving a two-variable problem. In part (a) we show the contours of the "six-hump camel" test function along with the rectangles and sampled points after 173 function evaluations. This function has has two global minima (i.e., both have the same optimal function value), and the sampled points cluster around them. Similarly, in part (b) we show the contours of the Branin test function after 195 function evaluations; in this case there are three global minima.

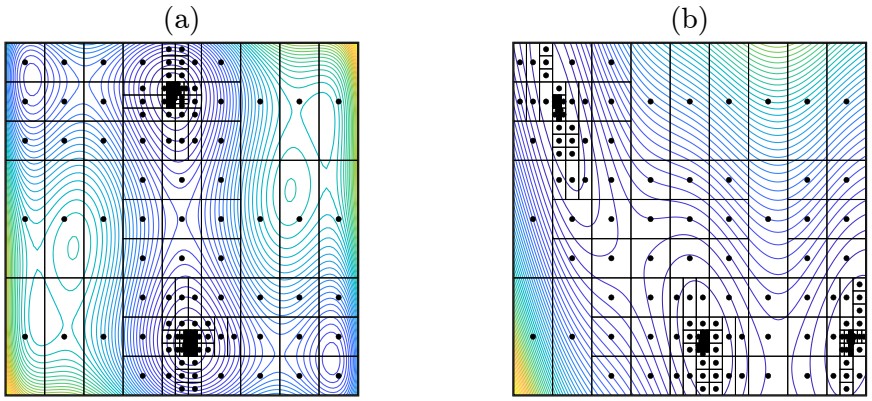

Figure 6: Two examples of DIRECT solving a two-variable test problem: (a) the six-hump camel function; (b) the Branin function.

3 Previous multiobjective extensions

Three multiobjective extensions of DIRECT appeared in the literature before *multi*DIRECT: nondominated sorting, hypervolume contribution, and multiobjective Pareto. The first two combine the multiple objectives into a single composite metric and then optimize this metric with single-objective DIRECT. The third method is a straightforward extension of DIRECT with Pareto selection to handle multiple objectives.

3.1 Nondominated sorting

In nondominated sorting, we first find the points that are nondominated in the output space (i.e., on the Pareto Front) and give these points a rank of 1. These points are then excluded from consideration. We then find the points that are nondominated among those that remain and give these points a rank of 2; we then also exclude these points from consideration. One continues in this way until no points are left. Figure 7 gives an example with two objectives, both of the lower-is-better variety, and nine points, four of which have rank 1, two have rank 2, and three have rank 3.

Wong et al. [12] extended DIRECT to multiple objectives by using the rank from nondominated sorting as the single objective in DIRECT. The intuition behind this approach is that the rank quantifies how good the point is from a multi-objective point of view. DIRECT decides whether to select and subdivide the rectangle based on the nondominated rank and the rectangle's size. In this way DIRECT will not only search near points on the Pareto front (rank equal to 1), but will also search near points away from the front (rank greater than 1), but which are still interesting due to their large rectangle size (i.e., high amount of unexplored territory and possibility for improvement). This seems very reasonable, and it works, but as we will see, it works less efficiently compared to the simDIRECT method we discuss later. One possible reason for the inefficiency is that the nondominated rank only imperfectly captures how close a point is to the Pareto front; for example, in Figure 1, two points have rank 2, yet one is much closer to the Pareto front than the other.

3.2 Hypervolume contribution

Another multiobjective extension of DIRECT, described in Wong et al. [12], is to use the "hypervolume contribution" of each point as the single objective in DIRECT. The hypervolume contribution is to be maximized, so in DIRECT we minimize -1 times the contribution. To understand the hypervolume contribution, one first needs to know the definition of the "dominated hypervolume," a metric commonly used in multiobjective optimization to track progress converging to the Pareto front.

Figure 8 illustrates the concepts of the dominated hypervolume and the hypervolume contribution, considering two objectives f_1 and f_2 of the lower-is-better variety. In Figure 8(a), point N is a "nadir point"; values of either f_1

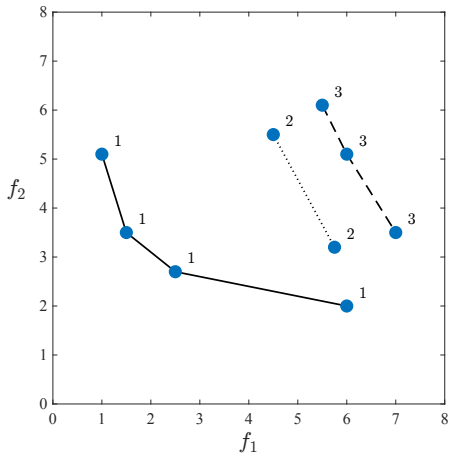

Figure 7: Nondominated sorting. The figure shows nine points output space of f_1 and f_2, both of the lower-is-better type. They are sorted into 3 layers of Pareto fronts. The number next to a dot is the rank.

or f_2 that are higher than those of the nadir point are unacceptable; thus, we are only interested in solutions that give outputs lying in the green shaded region below and to the left of N. In Figure 8(b) we show the same nine points as before and label the four nondominated ones as A, B, C, and D. Each of these points dominates anything above and to the right of it. The "dominated hypervolume" is the volume of the region that is dominated by one or more the points, but still better than the nadir point (shaded gray in Figure 8(b)). In the numerical results we present later, we compute the hypervolume using the open-source software from Foseca et al. [13], available at `http://lopez-ibanez.eu/hypervolume`.

Now suppose one of the points is deleted, such as point B in Figure 8(c). In this case, the dominated hypervolume will be reduced, and the magnitude of this reduction is the "hypervolume contribution" of point B. The contribution can also be visualized, as in Figure 8(d), as the volume of the region between the Pareto front with all the points and the Pareto front with all points except B (yellow region in Figure 8(d)). The hypervolume contribution will be low for nondominated points that are in crowded areas of the Pareto front, and high for points in less crowded regions. In this way, the hypervolume-contribution criterion favors sampling in less crowded regions of the Pareto front, leading to a more even distribution of points, and this is considered one of its strengths.

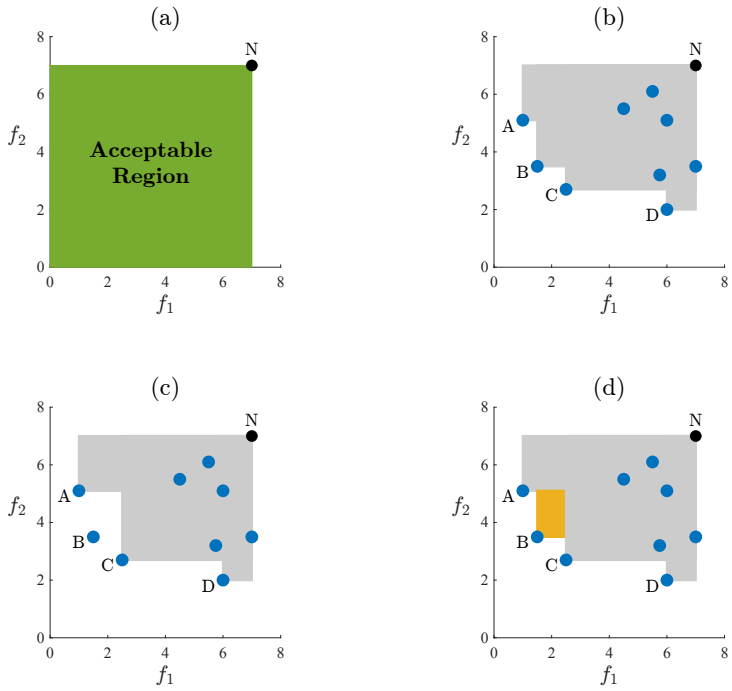

Figure 8: Hypervolume contribution. (a) the acceptable region lies below and to the left of the nadir point N; (b) the dominated hypervolume is the volume of the area dominated by the points on the Pareto front but better than the nadir point; (c) the dominated hypervolume with B deleted; (d) the hypervolume contribution of B in yellow.

Let's now compare the hypervolume approach with nondominated sorting. First, note that all dominated points have a hypervolume contribution of zero, since the dominated region doesn't change when they are deleted. Thus the hypervolume-contribution method lumps all the dominated points in one "bad" category, but makes distinctions between points on the Pareto front which will generally have different (positive) hypervolume contributions. Nondominated sorting is exactly the opposite: all points on the Pareto front are lumped into one "good" category (rank 1), while the dominated points will generally have several ranks (2,3,4, etc.) and so are distinguished from each other.

Because the hypervolume approach lumps all dominated points in one "bad" category (hypervolume zero), they are all tied for the objective, and so DIRECT will usually select just one of them (the one in the largest rectangle;

possibly more if several rectangles are tied for the biggest size). As a result, using the hypervolume-contribution as the objective in DIRECT may delay the discovery of parts of the Pareto front that initially are in rectangles whose center points are dominated.

3.3 Multiobjective Pareto selection

A third method for extending DIRECT to multiple objectives is to use multiobjective Pareto selection [14, 12]. In this approach, one selects the rectangle(s) that are nondominated, or Pareto optimal, with respect to all of the objectives (lower is better) and rectangle size (higher is better). This method doesn't aggregate the multiple objectives into a single composite, and it usually works better than nondominated sorting and hypervolume contribution. However, as in the single objective case, Pareto selection has the downside that it can temporarily get stuck in some area of the design space, wasting precious function evaluations pursuing extremely small improvements in the Pareto front. Being based only on relative values, the Pareto method can't distinguish small improvements from large ones. As we will see in the numerical results presented later, this can result in a nonuniform covering of the Pareto front, with scattered dense clusters of points in areas where the search got temporarily stuck.

4 From *multi*DIRECT to simDIRECT

In their *multi*DIRECT algorithm, Lovison and Miettinen [1] extend DIRECT to multiple objectives by adapting the concept of "potentially optimal rectangles" from one objective to multiple objectives. Among the extensions discussed so far, it is the most consistent with the original Lipschitzian motivation of DIRECT.

Recall that, in the single-objective case, we said that rectangle i is potentially optimal if there exists a Lipschitz constant $K > 0$ with the property that rectangle i has the lowest lower bound:

$$f(\mathbf{c}_i) - Kd_i \leq f(\mathbf{c}_j) - Kd_j, \quad \text{for all } j = 1, \ldots, n. \qquad \text{(2 revisited)}$$

and, in addition, this lower bound is better than the current best solution by some non-trivial amount $\epsilon > 0$:

$$f(\mathbf{c}_i) - Kd_i \leq f_{\min} - \epsilon. \qquad \text{(3 revisited)}$$

How does this change with $M > 1$ objectives? Well, we now have M Lipschitz constants $K_1, ..., K_M$ and rectangle i now has a *vector* of lower bounds:

$$(f_1(\mathbf{c}_i) - K_1 d_i, \;\; f_2(\mathbf{c}_i) - K_2 d_i, \;\ldots,\; f_M(\mathbf{c}_i) - K_M d_i)$$

Unfortunately, we can't require this vector of lower bounds to have the same property as in the single-objective case; that is, we can't say that "the vector of lower bounds for rectangle i is lower than the vector of lower bounds for any rectangle $j \neq i$." The reason is that, in the multiobjective case, the notion of "is lower than" is ill-defined: rectangle i can have a better bound than rectangle j on one objective and worse bound on another objective. What we *can* say, however, is that the vector of lower bounds for rectangle i *is not dominated by* the vector of lower bounds for any rectangle $j \neq i$.

In the single objective case, we also required the lower bound for rectangle i be less than or equal to $f_{\min} - \epsilon$; in other words, the lower bound for rectangle i had to be non-trivially better than the current best solution. In the multiobjective case, we can't say this because there is no single best solution $f_{\min}$; instead, the closest concept to "the solution" is the set of non-dominated sampled points. So what we can require is that the vector of lower bounds for rectangle i must *be nondominated by* the vector of outputs of any point p on the Pareto front. Furthermore, if we want it to be "nondominated by some nontrivial amount," we can require that the vector of lower bounds for rectangle i be nondominated by $(f_1(\mathbf{c}_p) - \epsilon_1, ..., f_M(\mathbf{c}_p) - \epsilon_M)$ for any point p on the Pareto front for small positive numbers $\epsilon_1, ..., \epsilon_M$.

Putting all of this together, Lovison and Miettinen give the following definition of what they call a "potentially *Pareto* optimal rectangle"[1]

[1]Lovison and Miettinen's definition is different from the one given here in two minor ways. First, they use the infinity norm in the definition of d_i instead of the 2-norm (see the discussion near equation (1)). Second, in equation (5), where we have $f_m(\mathbf{c}_p) - \epsilon_m$, they use $f_m(\mathbf{c}_p) - \epsilon|f_m(\mathbf{c}_p)|$; that is, the required improvement over Pareto point p is based on a common *relative* tolerance ϵ, as opposed to objective-specific *absolute* tolerances ϵ_m.

Definition 2. Potentially Pareto optimal hyperrectangle
Let M be the number of objectives; n be the number of rectangles; P be the set of rectangles with nondominated centerpoint outputs; and $\epsilon_m > 0$ be the desired accuracy for objective $m = 1, ..., M$. We say that rectangle i is *potentially Pareto optimal* if there exist positive constants $K_1, ..., K_M$ such that

$$\begin{gathered}(f_1(\mathbf{c}_i) - K_1 d_i,\ \ f_2(\mathbf{c}_i) - K_2 d_i,\ \ldots,\ f_M(\mathbf{c}_i) - K_M d_i) \\ \text{is not dominated by} \\ (f_1(\mathbf{c}_j) - K_1 d_j,\ \ f_2(\mathbf{c}_j) - K_2 d_j,\ \ldots,\ f_M(\mathbf{c}_j) - K_M d_j)\,, \\ \text{for all } j \in \{1, 2, ..., n\} \text{ with } j \neq i\end{gathered} \tag{4}$$

and, in addition,

$$\begin{gathered}(f_1(\mathbf{c}_i) - K_1 d_i,\ \ f_2(\mathbf{c}_i) - K_2 d_i,\ \ldots,\ f_M(\mathbf{c}_i) - K_M d_i) \\ \text{is not dominated by} \\ (f_1(\mathbf{c}_p) - \epsilon_1,\ \ f_2(\mathbf{c}_p) - \epsilon_2,\ \ldots,\ f_M(\mathbf{c}_p) - \epsilon_M)\,, \\ \text{for all } p \in P\end{gathered} \tag{5}$$

Note that this definition of "potentially Pareto optimal" reduces to the original definition of "potentially optimal" if the number of objectives is one. To see this, note that, if there is just one objective, then we are dealing with a single lower bound, $f_1(\mathbf{c}_i) - K_1 d_i$, not a vector, and the only way for it to be nondominated is for $f_1(\mathbf{c}_i) - K_1 d_i \leq f_1(\mathbf{c}_j) - K_1 d_j$ for all $j \neq i$, which is the same as our previous equation (2). Furthermore, with just one objective, the Pareto set P consists of the point(s) with the best objective function value, so $f_1(\mathbf{c}_p) = f_{\min}$. Equation (5) then reduces to $f_1(\mathbf{c}_i) - K_1 d_i \leq f_{\min} - \epsilon_1$, which is the same as our previous equation (3).

In hindsight, the concept of "potentially Pareto optimal rectangle" is clearly the most natural way to extend DIRECT to handle multiple objectives. So why was it not proposed sooner? The most likely reason is that implementing the concept is extremely difficult. In the case of one objective, we are able to visualize how to do the selection with our two-dimensional f-d diagram (Figure 4). With $M \geq 2$ objectives, the visualization must be done in three or more dimensions, and the geometric reasoning used to identify the potentially Pareto optimal rectangles is extremely complex, involving such concepts as "dominance cones," "virtual Pareto front," and "tipping vectors."

Inspired by *multi*DIRECT, but unable to fully understand the numerical implementation, I looked for a way to simplify the method while maintaining the basic idea. The idea I finally came up with was as follows. Why not keep track of the average absolute rate of change for each objective m, denoted R_m (more on how to do this later) and then, in the definition of "potentially Pareto optimal rectangle," require that $K_m = \alpha R_m$ for some positive scale factor α? That is, why not say rectangle is i "potentially Pareto optimal" if there exists a scale factor $\alpha > 0$ such that conditions (4) and (5) are satisfied if we use $K_m = \alpha R_m$? With this simplification, we reduce an M-dimensional search over $K_1, ..., K_M$ to a much easier one-dimensional search over possible values of α. I call this simplified version "simDIRECT," short for "**si**mplified **m**ultiobjective **DIRECT**."

It turns out to be easy to estimate the "average absolute rate of change for each objective $m = 1, ..., M$." Each time a rectangle is trisected, we can compute the absolute value of the *rate of change* of $f_m(\cdot)$ between the center of the "parent" rectangle and the center of the "child" rectangle:

$$\text{absolute rate of change} = \frac{|f_m(\mathbf{c}_{\text{parent}}) - f_m(\mathbf{c}_{\text{child}})|}{\|\mathbf{c}_{\text{parent}} - \mathbf{c}_{\text{child}}\|} \tag{6}$$

For each trisection, we get two absolute rates of change, one for the left child and one for the right child. Averaging these rates of change over all trisections, we can calculate an "average absolute rate of change." As a detail, if the average rate of change so computed is zero, we replace it with a small number (e.g., 10^{-10}) to avoid dividing by zero in later calculations.

Any rectangle selected by simDIRECT is also selected by multiDIRECT, but not vice versa. Why? Well, if rectangle i is selected by simDIRECT, then there exists an $\alpha > 0$ such that conditions (4) and (5) are satisfied if we use $K_m = \alpha R_m$. Since *multi*DIRECT only requires these conditions to be satisfied using *some* positive $K_1, ..., K_M$ values, the ones used by simDIRECT suffice, and so *multi*DIRECT would also select rectangle i.

Thought of another way, simDIRECT selects any rectangle i for which conditions (4) and (5) can be satisfied with K_m values proportional to the R_m values. In contrast, *multi*DIRECT selects all these rectangles *plus* those where the conditions can only be satisfied with K_m values that are *not* proportional to the R_m values. Whether *multi*DIRECT performs better or worse than simDIRECT therefore depends upon whether selecting these additional rectangles helps discover the optimum sooner, or whether it simply adds extra overhead.

5 Implementing simDIRECT

The simDIRECT algorithm follows exactly the same steps shown for DIRECT, as described earlier, with the following changes:

1. For each search point we evaluate M functions (instead of one).

2. After evaluating new points in an iteration, we update the set P of rectangles whose centerpoint output vectos are on the Pareto front (instead of updating $\mathbf{x}_{\min}$ and $f_{\min}$).

3. After we trisect a rectangle, we compute the rates of change of each objective function between the parent and child centerpoints, and use this to update the average absolute rates of change R_m for $m = 1, ..., M$

4. In each iteration we select the set of potentially *Pareto* optimal rectangles, where in the definition of "potentially Pareto optimal" we restrict $K_m = \alpha R_m$ for some $\alpha > 0$.

As changes 1-3 are straightforward, we will focus here on change 4, rectangle selection.

In rectangle selection, we will cycle through rectangles $i = 1, .., n$, determining whether the rectangle is "potentially Pareto optimal," that is, whether there exists an $\alpha > 0$ such that conditions (4) and (5) are satisfied if we set $K_m = \alpha R_m$. Our strategy will not be to look for the alpha values that satisfy these conditions, but rather to start with all possible alpha values — that is, the interval $[0, \infty]$ — and then subtract alpha intervals of the form $[a, b]$ that we can show do *not* meet one of the conditions. Once we have made all possible subtractions, we then examine what is left. If some part of the original alpha interval remains, then we will know that there exists some alpha that meet the conditions and, hence, rectangle i is indeed potentially Pareto optimal; otherwise, rectangle i is not potentially Pareto optimal.

To implement this strategy in code, it is helpful to write a program that can do this interval subtraction. Sometimes subtraction is simple. For example, if we start with $[0, \infty]$ and subract $(9, \infty]$ we get $[0, 9]$. If we then subtract $[0, 2)$, we are left with $[2, 9]$. It can a little tricky if the interval being subtracted lies inside an existing interval. For example, if we continue and and subtract $(4, 5)$ we are left with *two* intervals $[2, 4]$ and $[5, 9]$. Because subtraction can increase the number of intervals, the computer program needs to store the current state as a *set of intervals* and, when given some interval

to subtract, it must go through every interval in the set and subtract it. Of course, after subtraction, it is possible that nothing will remain, and the code should then report that there are no intervals in the set. Clearly, this can be done with careful coding and we will not detail it further.

Now suppose we are tying to determine whether rectangle i is "potentially Pareto optimal," that is, whether it satisfies conditions (4) and (5) using $K_m = \alpha R_m$ and some $\alpha \geq 0$. We begin by initializing the set of α values that work to $[0, \infty]$ and then subtract intervals of α values for which either condition (4) or (5) does *not* hold.

Let's start with condition (4)). This condition will *not* be true if, for some $j \neq i$, the vector of lower bounds for rectangle j dominates the vector of lower bounds for rectangle i, that is, if

$$f_m(\mathbf{c}_j) - \alpha R_m d_j \leq f_m(\mathbf{c}_i) - \alpha R_m d_i, \text{ for all } m = 1, ..., M \tag{7}$$

with one of the inequalities being strict. Any alpha values that make this true can be subtracted. To identify these values, we will consider the cases $d_j = d_i$, $d_j > d_i$, and $d_j < d_i$ separately.

If $d_j = d_i$, then equation (7) reduces to $f_m(\mathbf{c}_j) \leq f_m(\mathbf{c}_i)$ for all m, with one inequality strict, which means that the outputs at the center of rectangle j dominate those for rectangle i. If this happens, then we can immediately say that rectangle j dominates rectangle i for all values of α, so rectangle i is *not* potentially Pareto optimal. Otherwise, we can say nothing and just move on to consider the next $j \neq i$.

Now consider the case $d_j > d_i$. In this case, we can rearrange equation (7) and, using our assumption that $d_j - d_i > 0$, obtain:

$$\alpha \geq \underset{m \in \{1,2,...,M\}}{\text{maximum}} \frac{f_m(\mathbf{c}_j) - f_m(\mathbf{c}_i)}{R_m(d_j - d_i)} \equiv a_{ij} \tag{8}$$

In this case we can subtract the interval $(a_{ij}, \infty]$ from our candidate alpha values, since for all these values rectangle j will dominate rectangle i. We subtract the open interval because we need $\alpha > a_{ij}$ to ensure that rectangle j is strictly better than rectangle i on some output.

Finally, let's consider the case $d_j < d_i$. In this case, we can rearrange terms in equation (7) and, using our assumption that $d_i - d_j > 0$, obtain

$$\alpha \leq \underset{m \in \{1,2,...,M\}}{\text{minimum}} \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_j)}{R_m(d_i - d_j)} \equiv b_{ij} \tag{9}$$

If $b_{ij} > 0$, then in this case we can subtract the interval $[0, b_{ij})$ from our candidate alpha values, since for all these values of α, rectangle j will dominate rectangle i.

If, after cycling through all rectangles $j \neq i$, some valid alpha values remain, then we need to move on to the second condition in the definition of "potentially Pareto optimal" given in equation (5), and consider whether this condition eliminates any alpha values. For equation (5) to *not* hold, we require that there exist some $p \in P$ such that

$$f_m(\mathbf{c}_p) - \epsilon_m \leq f_m(\mathbf{c}_i) - \alpha R_m d_i, \text{ for all } m = 1, ..., M \tag{10}$$

with one inequality being strict. Rearranging terms to isolate α, we get

$$\alpha \leq \underset{m \in \{1,2,...,M\}}{\text{minimum}} \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_p) + \epsilon_m}{R_m d_i} \equiv b_{ip} \tag{11}$$

For each $p \in P$, if $b_{ip} > 0$, then we can subtract $[0, b_{ip})$ from our candidate alpha values, since for all these values the vector of lower bounds for rectangle i would not improve non-trivially (i.e., by at least $(\epsilon_1, ..., \epsilon_M)$) over Pareto point p.

At this point we will have considered all the ways the two conditions for potential Pareto optimality could *not* be true for rectangle i, and we will have subtracted all the associated alpha intervals. Thus, if any alpha values remain, rectangle i is potentially Pareto optimal and should be selected.

Putting everything together, we summarize the "simDIRECT Rectangle Selection Algorithm" with the pseudocode shown in Algorithm 1. Finally, in Algorithm 2 we give pseudocode for simDIRECT itself.

Algorithm 1 simDIRECT Rectangle Selection

Input: Number rectangles n, number objectives M, objective values at rectangle centerpoints $f_m(\mathbf{c}_i)$, rectangle center-vertex distances d_i, average absolute rates of change of objectives R_m, desired accuracy of objectives ϵ_m; set P of rectangles whose centerpoint outputs are nondominated.

Output: Set of selected rectangles.

If $n = 1$, select this single rectangle; otherwise, continue as follows.
Initialize the set of selected rectangles to be the empty set
for $i = 1, ..., n$ **do**
  $\mathcal{S} = \{[0, \infty]\}$ ▷ $\mathcal{S}$ is the set of α intervals satisfying conditions (4) and (5)
  for $j = 1, ..., n$ and $j \neq i$ **do**
    if $d_j = d_i$ **then**
      if $f_m(\mathbf{c}_j) \leq f_m(\mathbf{c}_i)\ \forall\ m$, with at least 1 ineq. strict **then** $\mathcal{S} = \emptyset$
    else if $d_j > d_i$ **then**
      Substract $(a_{ij}, \infty]$ from $\mathcal{S}$ where $a_{ij} = \max_m \frac{f_m(\mathbf{c}_j) - f_m(\mathbf{c}_i)}{R_m(d_j - d_i)}$
    else
      if $b_{ij} \equiv \min_m \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_j)}{R_m(d_i - d_j)} > 0$ **then** subtract $[0, b_{ij})$ from $\mathcal{S}$
    end if
  end for
  for each p in P **do**
    if $b_{ip} \equiv \min_m \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_p) + \epsilon_m}{R_m d_i} > 0$ **then** subtract $[0, b_{ip})$ from S
  end for
  if $\mathcal{S} \neq \emptyset$ **then** add rectangle i to the set of selected rectangles
end for

6 Numerical results without constraints

In this section, we will explore the performance of all the multiobjective extensions of DIRECT we've discussed so far using two test problems. Both problems assume lower and upper bounds on the variables, but otherwise are unconstrained; we will add general nonlinear constraints to simDIRECT in a later section. Our goal here is to get a flavor of how the different extensions perform, how they scale with the number of variables and objectives, and how they compare with the multiobjective optimization routines that come with MATLAB. We leave more comprehensive comparisons and evaluations to future work.

Algorithm 2 simDIRECT

Input: Objective function $f(\cdot)$; number of objectives M; lower and upper bounds on the variables; iteration limit $I_{\max}$; desired accuracy for the objectives $\epsilon_m > 0$

Output: All sampled points, with a subset P that approximates the global Pareto front for the M objectives in both the input and output spaces

Normalize the design space to the unit hypercube.
Evaluate the outputs at the hypercube center $\mathbf{c}_1$; initialize the set $P = \{1\}$
Initialize $t(\ell) = 0$ for $\ell = 1, \ldots, d$ ▷ $t(\ell)$ counts no. times trisected on dimension ℓ
Initialize sums and counters for computing ave. rates of change R_m.
Initialize the iteration counter I to zero
while $I < I_{\max}$ **do**
  Use Algo. 1 to find the set $\mathcal{O}$ of potentially Pareto optimal rectangles
  for each hyperrectangle in $\mathcal{O}$ **do** ▷ process $\mathcal{O}$ from low to high evaluation no.
    Find the set $\mathcal{L}$ of dimensions with maximum side length
    Select ℓ^* in $\mathcal{L}$ with the lowest $t(\ell)$, breaking ties in favor of lower ℓ
    Trisect the rectangle along dimension ℓ^* and update the partition
    Evaluate the center points of the outer two hyperrectangles
    Update the ave. rates of change R_m based on the new evaluations
    Increment $t(\ell^*)$
  end for
  Update the Pareto set P based on the new evaluations
  Increment I
end while

6.1 The $L\&H_{2\times2}$ problem

The first test problem is called $L\&H_{2\times2}$ and was used by Lovison and Miettinen in their paper on *multi*DIRECT. The problem was originally given in [15] and has two objectives and two variables. In the original paper the objectives are to be maximized, so we multiplied them by -1 since we assume we are minimizing. We have also simplified the original formulation to get expressions that are equivalent but more compact:

$L\&H_{2\times2}$ problem:

$\underset{x_1,x_2}{\text{minimize}}$	$(\ f_1(x_1,x_2),\ f_2(x_1,x_2)\ )$
subject to	
	$-0.75 \le x_1 \le 0.75$
	$-2.50 \le x_2 \le 0.12$
where	
	$f_1(x_1,x_2) = -\frac{\sqrt{2}}{2}x_1 + \sqrt{\frac{4\pi}{65}}\exp\left(-\frac{x_1^2+x_2^2}{0.4225}\right) + \sqrt{\frac{90\pi}{112}}\exp\left(-\frac{x_1^2+(x_2+1.5)^2}{7.84}\right)$
	$f_2(x_1,x_2) = +\frac{\sqrt{2}}{2}x_1 + \sqrt{\frac{4\pi}{65}}\exp\left(-\frac{x_1^2+x_2^2}{0.4225}\right) + \sqrt{\frac{90\pi}{112}}\exp\left(-\frac{x_1^2+(x_2+1.5)^2}{7.84}\right)$
Nadir point:	
	$[-0.8, -0.8]$
Hypervolume:	
	1.11525

Figure 9 shows the known solution to this problem (this solution was obtained with simDIRECT using $50,000$ function evaluations). In this figure, part (a) shows the Pareto front in output space, and part (b) shows the Pareto front in input space. If we use $-0.8, -0.8$ as the nadir point, then the optimal dominated hypervolume is 1.11525.

Figures 10 and 11 show the results after 500 function evaluations on the $L\&H_{2\times2}$ problem using all four previously-discussed multiobjective extensions of DIRECT (simDIRECT used $\epsilon_1 = \epsilon_2 = 10^{-4}$). Figure 10 shows the output space, and Figure 11 shows the input space. In both figures, we use red circles for the nondominated points and blue "+" signs for all other sampled points.

Looking at the results for the output space in Figure 10, all methods approximate the true Pareto front fairly well, but simDIRECT's search finds more points on the Pareto front.

The situation is different for the input space shown in Figure 11. In particular, the two methods that combine the outputs into a single com-

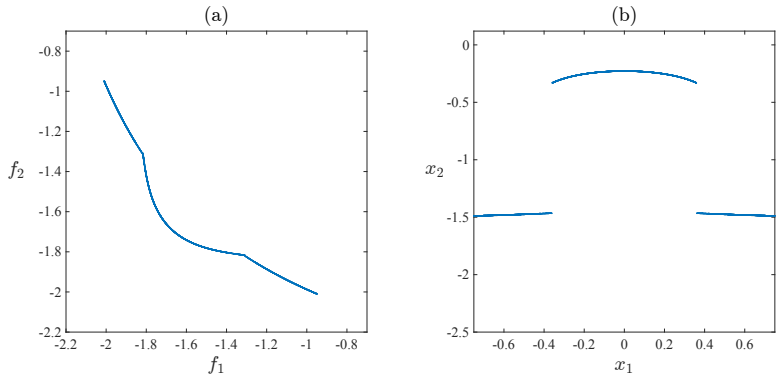

Figure 9: The Pareto front of the $L\&H_{2\times 2}$ problem: (a) output space; (b) input space.

posite objective — nondominated sorting and hypervolume contribution — sample almost uniformly with little focus around the Pareto front. The multiobjective Pareto method does better, but the sampled points in the input space are not as uniformly distributed on the Pareto front as the points from simDIRECT. The clusters of points in the plot for the multiobjective Pareto method correspond to areas where the method got "stuck" chasing very small improvements — the key weakness of that approach.

To compare all four methods in one chart, in Figure 12 we plot the "hypervolume fractional gap" versus the number of function evaluations, where

$$\text{gap}(t) \equiv 1 - \frac{\text{hypervolume}(t)}{\text{optimal hypervolume}} \tag{12}$$

and where t denotes the number of function evaluations. Intutively, gap(t) is the fraction of the true dominated hypervolume that remains to be discovered after t function evaluations; it starts near one and, hopefully, decreases to zero during the optimization. Looking at Figure 12, we see that all the methods make good initial progress, but simDIRECT outperforms them after about 50 function evaluations.

To put the performance of simDIRECT in perspective, Figures 13-15 compare simDIRECT to Matlab's "gamultiobj" and "paretosearch." Matlab's "gamultiobj" uses a variant of the well-known NSGA-II genetic algorithm [2]. Matlab's "paretosearch" uses pattern search [16]. Looking at Figure 13, we see that all three algorithms get close to the Pareto front in output space. However, from Figure 14, we see that the Matlab routines are

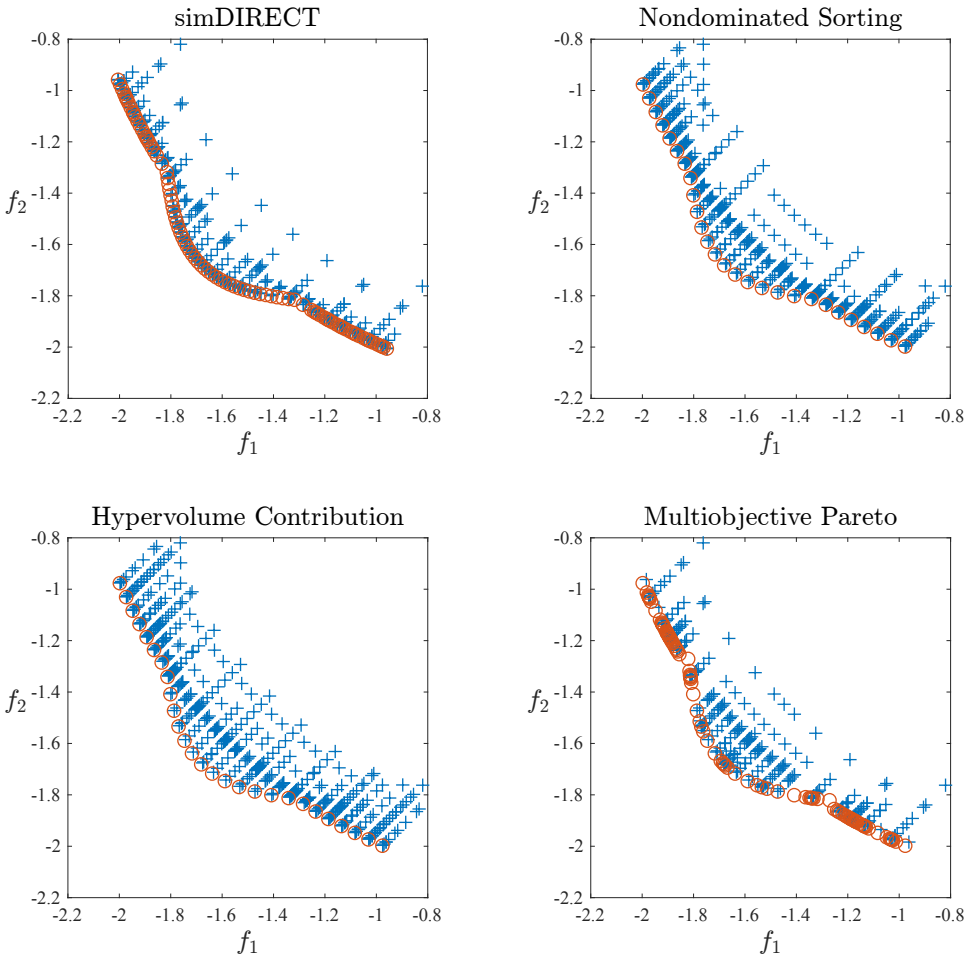

Figure 10: Sampled points in output space from running different multi-objective extensions of DIRECT on the $L\&H_{2\times 2}$ problem for 500 function evaluations. Red circles are nondominated points, blue "+" signs are dominated points.

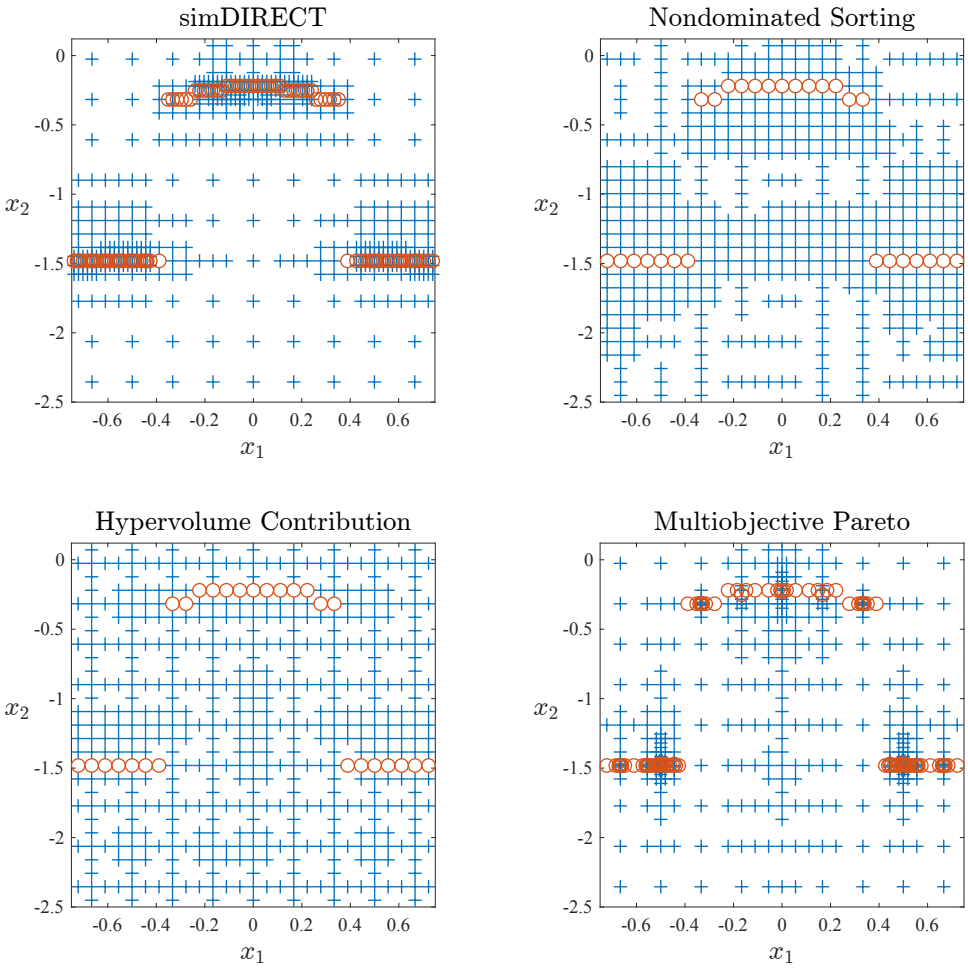

Figure 11: Sampled points in input space from running different multiobjective extensions of DIRECT on the $L\&H_{2\times2}$ problem for 500 function evaluations. Red circles are nondominated points, blue "+" signs are dominated points.

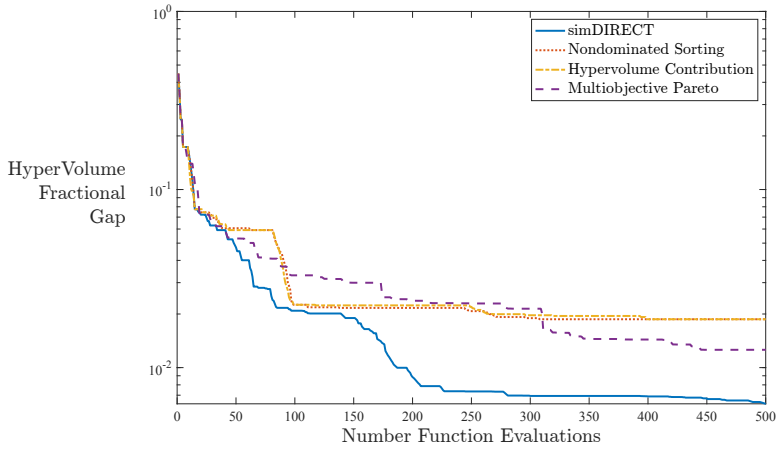

Figure 12: Convergence histories for different multiobjective extensions of DIRECT on the $L\&H_{2\times 2}$ problem for 500 function evaluations.

much worse than simDIRECT at identifying the Pareto front *in the input space*, and Figure 15 shows that simDIRECT exhibits better convergence in terms of the hypervolume fractional gap.

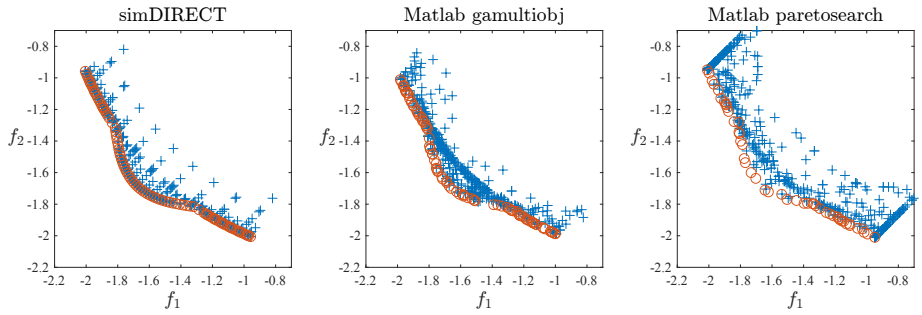

Figure 13: Sampled points in output space after 500 function evaluations on the $L\&H_{2\times2}$ problem for simDIRECT and Matlab's gamultiobj and paretosearch.

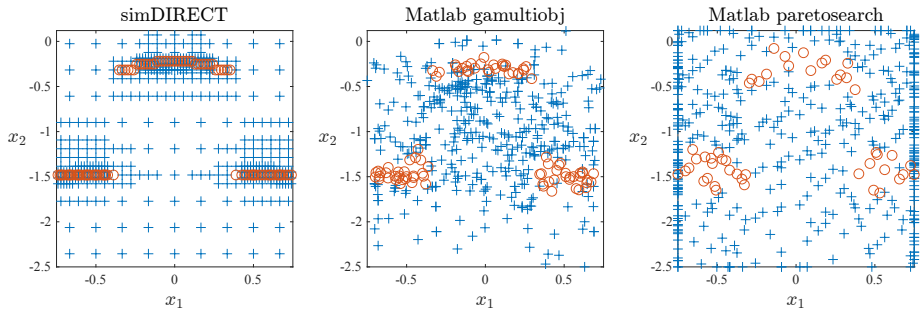

Figure 14: Sampled points in input space after 500 function evaluations on the $L\&H_{2\times2}$ problem for simDIRECT and Matlab's gamultiobj and paretosearch.

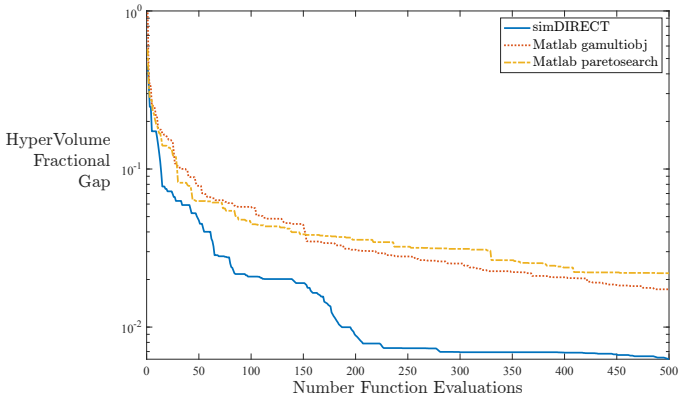

Figure 15: Convergence of the hypervolume fractional gap on the $L\&H_{2\times2}$ problem for simDIRECT and Matlab's gamultiobj and paretosearch.

6.2 The DTLZ2 problem

The second test problem we will consider is the DTLZ2 problem from Deb et al. [17]. This problem is constructed to have a known Pareto front and to be adjustable in terms of the number of objectives and variables. Here we will use it to explore how well simDIRECT scales with problem size, considering 2-3 objectives and 2-16 variables. Let's start with the two-objective version:

DTLZ2 problem with two objectives and d variables

minimize $\underset{x_1,...,x_d}{}$ $(f_1(x_1, ..., x_d),\ f_2(x_1, ..., x_d))$

subject to

$$0 \le x_\ell \le 1, \quad \ell = 1, ..., d$$

where

$$f_1(x_1, ..., x_d) = \left(1 + \sum_{\ell=2}^{d} (x_\ell - x^*)^2\right) \cos\left(\tfrac{\pi}{2} x_1\right)$$
$$f_2(x_1, ..., x_d) = \left(1 + \sum_{\ell=2}^{d} (x_\ell - x*)^2\right) \sin\left(\tfrac{\pi}{2} x_1\right)$$

Options for x^*:

0.5	– as in Deb, but too easy for simDIRECT
$\sqrt{2}/2$	– neutral for simDIRECT
$1/3$	– hard for simDIRECT

Pareto front:

$$x_1 \in [0, 1]$$
$$x_\ell = x^*, \quad \ell = 2, ..., d$$

Nadir point:

$$[1.5, 1.5]$$

Hypervolume:

$$2.25 - \tfrac{\pi}{4}$$

The DTLZ2 problem is constructed so that the Pareto front is the unit circle in the first quadrant of (f_1, f_2)-space. The inputs corresponding to this Pareto front have the form $(x_1, x^*, x^*, ..., x^*)$ where x_1 varies over $[0, 1]$ and the remaining $d - 1$ elements are all fixed at x^*. In the original paper, the authors use $x^* = 0.5$, but this gives simDIRECT an unrealistic advantage because simDIRECT begins by sampling the midpoint of the domain, which is $(0.5, ..., 0.5)$, so it immediately gets $d - 1$ of the inputs correct for free. To be more realistic, we will explore DTLZ2 with two other values for x^*.

The first is $x^* = \sqrt{2}/2 \approx 0.7071$, which is neutral for simDIRECT, giving no special advantage or disadvantage. The second is $x^* = 1/3$, which is the worst case for simDIRECT, as it can only be approached asymptotically and, being on the border of several rectangles, it will be approached redundantly from many directions.

Figure 16 shows the sampled points in output space after running simDIRECT for 5000 function evaluations on DTLZ2 with $x^* = \sqrt{2}/2$. As before, red circles are nondominated points and blue "+" signs are dominated points. The Pareto front is well approximated for 2-12 variables, and somewhat less well approximated with 14-16 variables.

Now consider Figure 17 which shows DIRECT's performance using the worst-case value of $x^* = 1/3$. As expected, the results are indeed worse, but the Pareto front is still recognizable up to 8 variables.

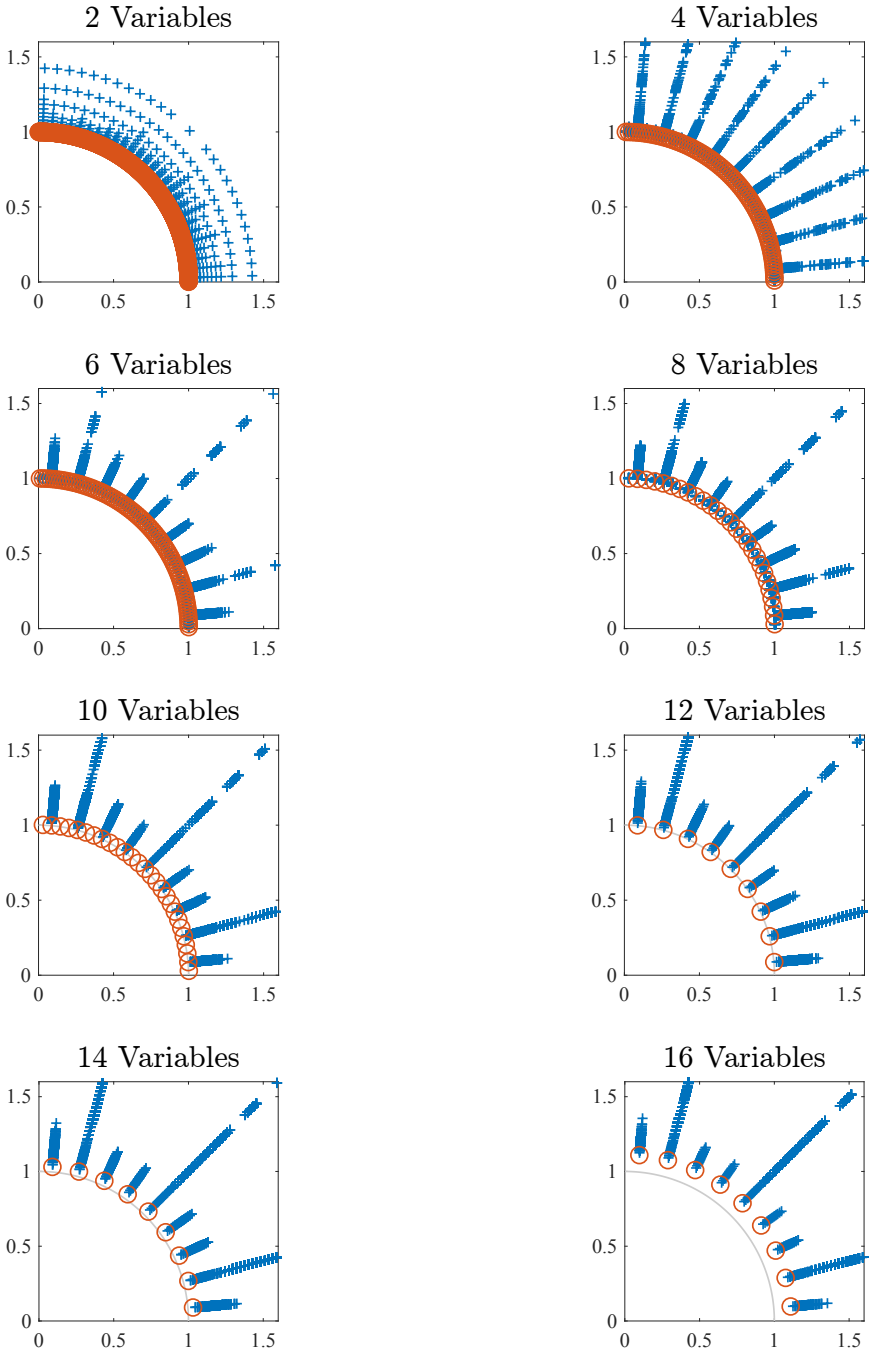

Figure 16: Results from running simDIRECT for 5000 function evaluations on the two-objective DTLZ2 problem with $x* = \sqrt{2}/2$.

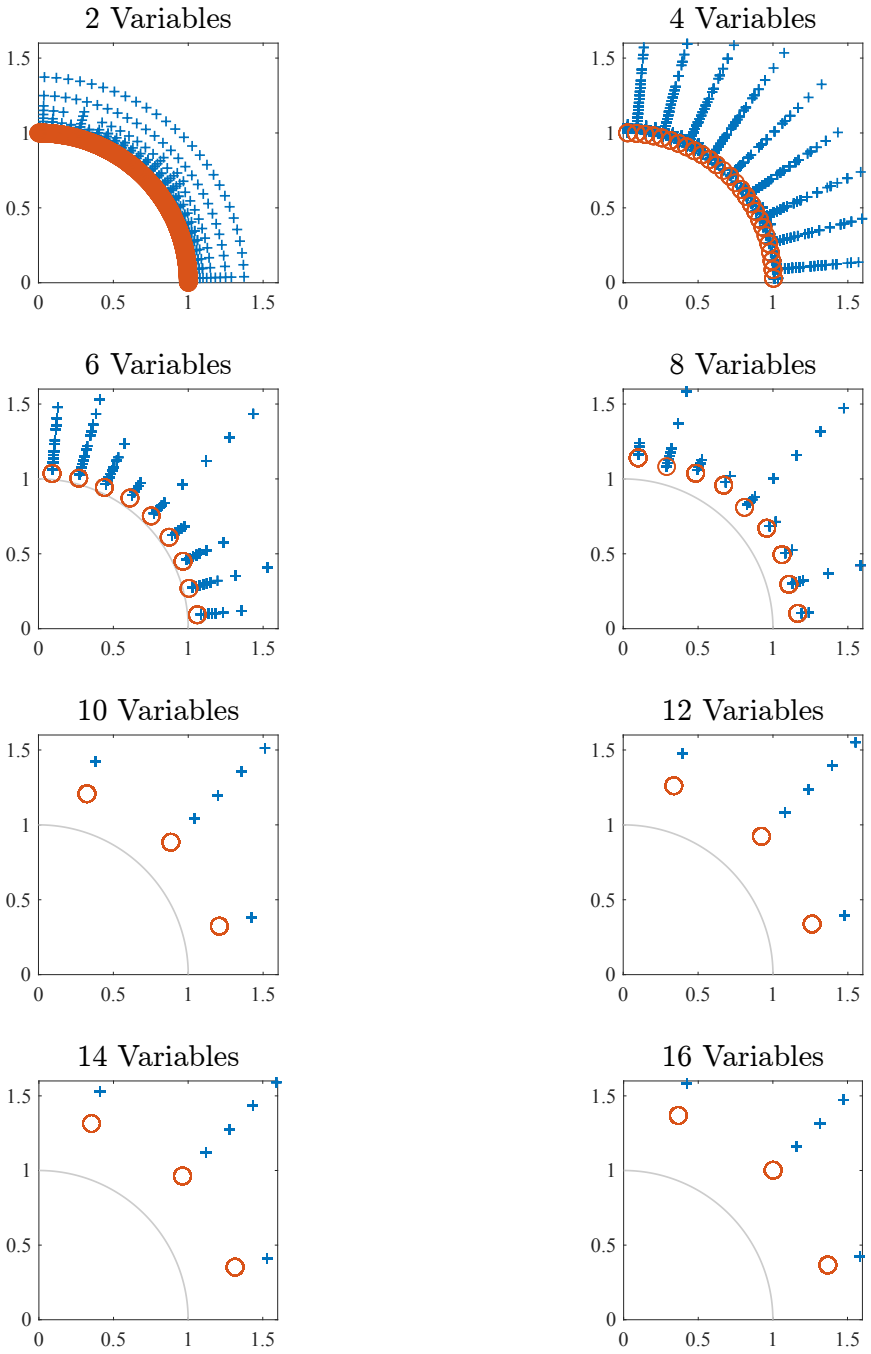

Figure 17: Results from running simDIRECT for 5000 function evaluations on the two-objective DTLZ2 problem with $x* = 1/3$.

To put the performance of simDIRECT in perspective, we also ran Matlab's "gamultiobj" on the DTLZ2 problem with $x* = \sqrt{2}/2$, again using 5000 function evaluations. Because gamultiobj is a stochastic algorithm, we made 21 runs starting with different random seed numbers (we used the 21 two-digit prime numbers). The results are shown in Figure 18, where for gamultiobj we show the range of results over the 21 runs. After 5000 evaluations, the results for gamultiobj are excellent, with the Pareto front clearly identified even with 16 variables. Furthermore, the hypervolume fractional gap after 5000 evaluations is much lower with gamultiobj than it is with simDIRECT for the 10- and 16-variable version of DTLZ2.

However, 5000 function evaluations is quite a lot for a small problem. If we focus on the relative performance over the first 600 runs, we find a diffent story: simDIRECT's performance is comparable to gamultiobj and, in the 16-variable case, actually a little better. The reason is simple: the genetic algorithm starts with an initial population with random points. In the case of gamultiobj, the default population size is 50 for up to five variable and 200 for more than five. So when running with 10 variables, gamultiobj spends 200 evaluations just evaluating the initial population and, with a limit of 600 evaluations, can only execute two additional generations. That's not a lot of iterations. In contrast, simDIRECT starts with one evaluation and the initial iterations have only a few points. For this reason, simDIRECT makes good initial progress. Later iterations of simDIRECT can have many (even over 1000) evaluations and, as a result, simDIRECT tends to make slower progress after an initial good start.

After just 600 function evaluations, simDIRECT's performance – while not as good as gamultiobj after 5000 evaluations – is still not bad, as can be seen in Figure 19. It roughly identifies the Pareto front with up to 12 variables. Thus, simDIRECT is indeed a good alternative to gamultiobj if one has a limited run budget (say, no more than 600), two objectives, and a small number of variables (say, up to 12).

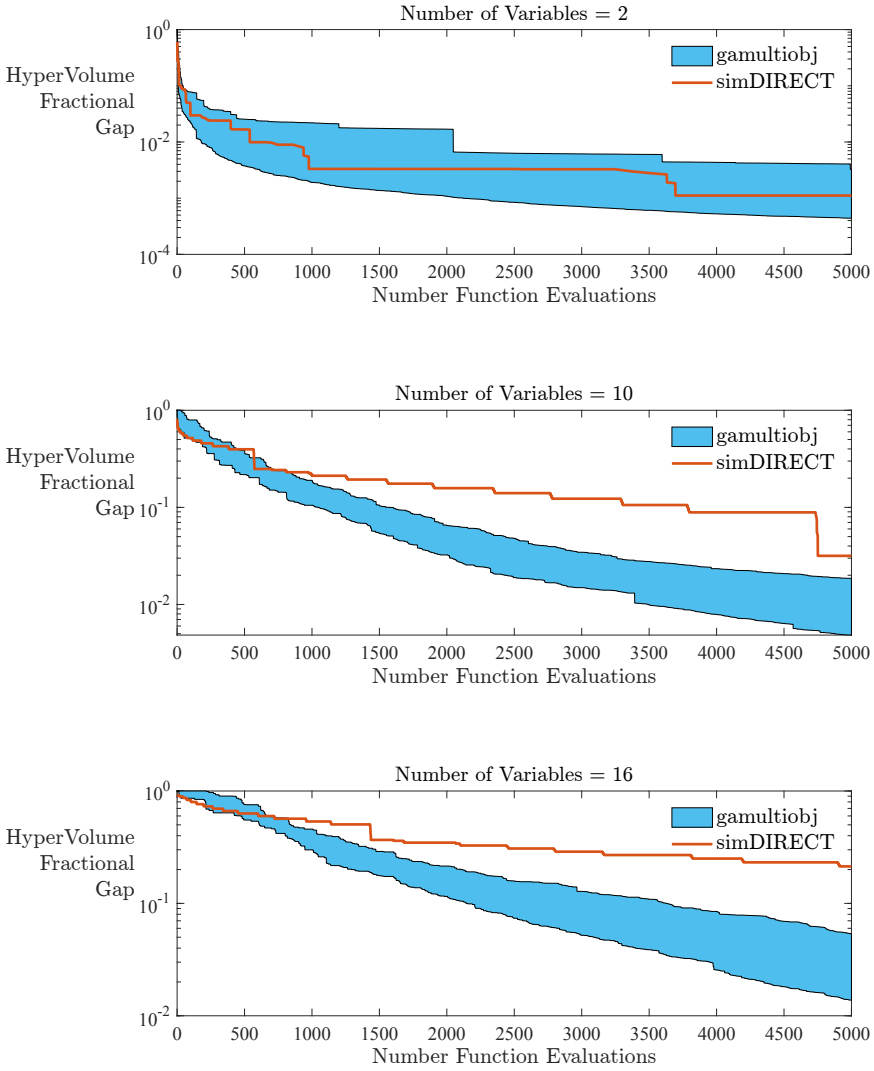

Figure 18: Comparing simDIRECT to gamultiobj in terms of progress in reducing the hypervolume fractional gap on the two-objective DTLZ2 problem with $x* = \sqrt{2}/2$. The blue band is the range from 21 runs of gamultiobj with different random seed numbers.

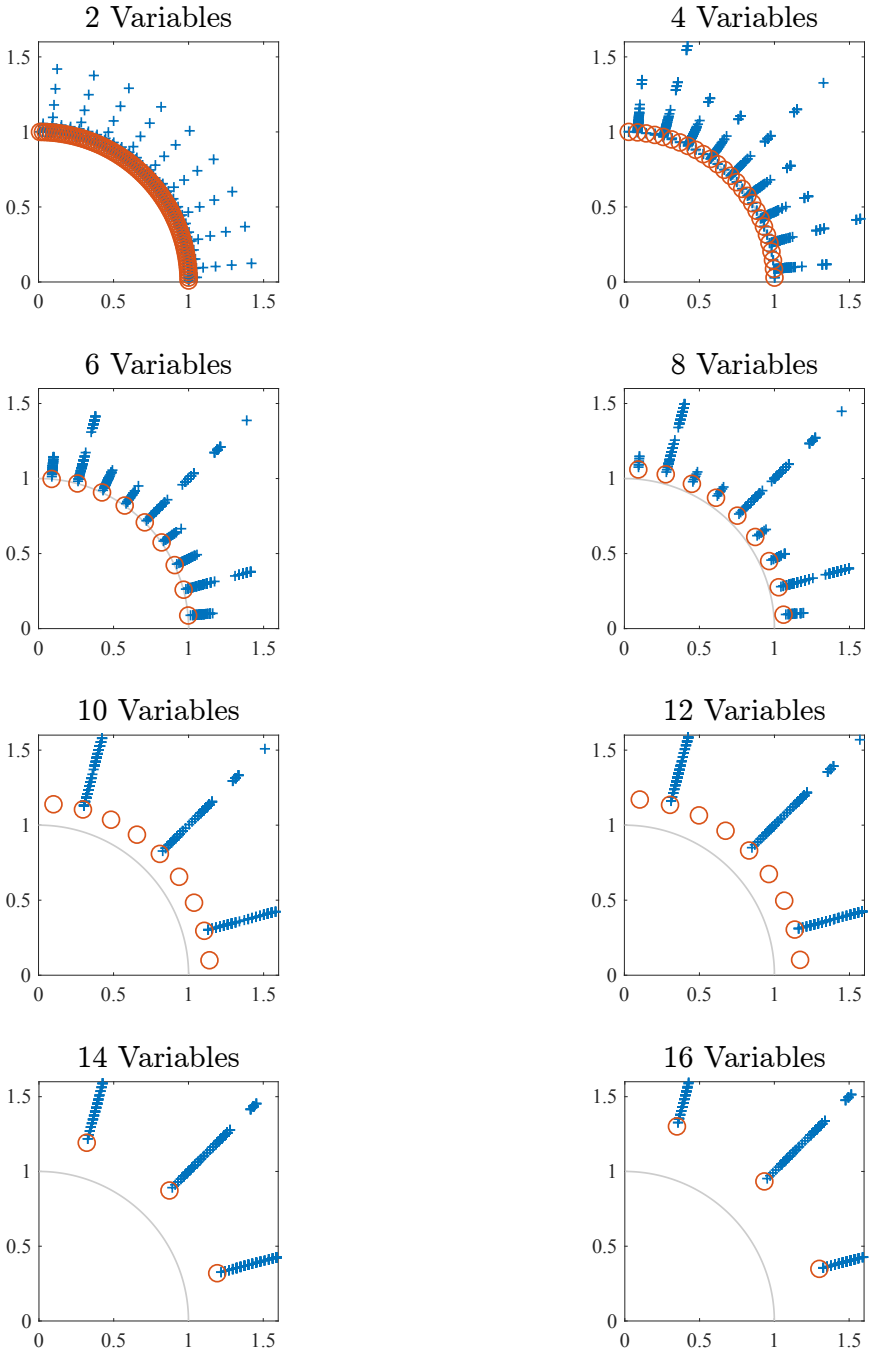

Figure 19: Samped points after 600 function evaluations of simDIRECT on the two-objective DTLZ2 problem with $x* = \sqrt{2}/2$.

Now let us turn our attention to the version of DTLZ2 with *three objectives*. In the three-objective problem, the Pareto set in output space is the surface of the unit sphere in the first octant (i.e., where $f_1, f_2, f_3 \geq 0$ and satisfy $f_1^2 + f_2^2 + f_3^2 = 1$). In input space, the Pareto set consists of vectors $(x_1, x_2, x^*, ..., x^*)$ where $x_1, x_2 \in [0, 1]$ and x^* is a constant. As mentioned before, Deb et al. [17] use $x^* = 0.5$, but this is too easy for DIRECT, so we will use $x^* = \sqrt{2}/2$. The problem can be stated formally as follows:

DTLZ2 problem with three objectives and d variables

$\underset{x_1,...,x_d}{\text{minimize}}$	$(\ f_1(x_1, ..., x_d),\ f_2(x_1, ..., x_d),\ f_3(x_1, ..., x_d)\ )$
subject to	
	$0 \leq x_\ell \leq 1, \quad \ell = 1, ..., d$
where	
	$f_1(x_1, ..., x_d) = \left(1 + \sum_{\ell=3}^{d}(x_\ell - x^*)^2\right) \cos\left(\frac{\pi}{2}x_1\right) \cos\left(\frac{\pi}{2}x_2\right)$
	$f_2(x_1, ..., x_d) = \left(1 + \sum_{\ell=3}^{d}(x_\ell - x^*)^2\right) \cos\left(\frac{\pi}{2}x_1\right) \sin\left(\frac{\pi}{2}x_2\right)$
	$f_3(x_1, ..., x_d) = \left(1 + \sum_{\ell=3}^{d}(x_\ell - x^*)^2\right) \sin\left(\frac{\pi}{2}x_1\right)$
Options for x^*:	
	0.5 – as in Deb, but too easy for simDIRECT
	$\sqrt{2}/2$ – neutral for simDIRECT
	$1/3$ – hard for simDIRECT
Pareto front:	
	$x_1, x_2 \in [0, 1]$
	$x_\ell = x^*, \quad \ell = 3, ..., d$
Nadir point:	
	$[1.5, 1.5, 1.5]$
Hypervolume:	
	$3.375 - \frac{\pi}{6}$

Surprisingly, with three objectives, simDIRECT's performance relative to Matlab actually *improves*, as shown in Figure 20. The improvement is most striking with the 16-variable version of the DTLZ2 problem. In the two-objective case, simDIRECT's hypervolume gap at 5000 evaluations was an order of magnitude worse than Matlab; now, in the three-objective case, simDIRECT's performance falls inside the range of results from MATLAB.

Moreover, as shown in Figure 21, simDIRECT captures the Pareto front closely for up to 10 variables and slightly less closely for 12-16 variables.

In summary, increasing the number of objectives from two to three led to poorer solutions (higher hypervolume gap) for both simDIRECT and Matlab's gamultiobj, but it appears that the performance of Matlab's genetic algorithm degraded more.

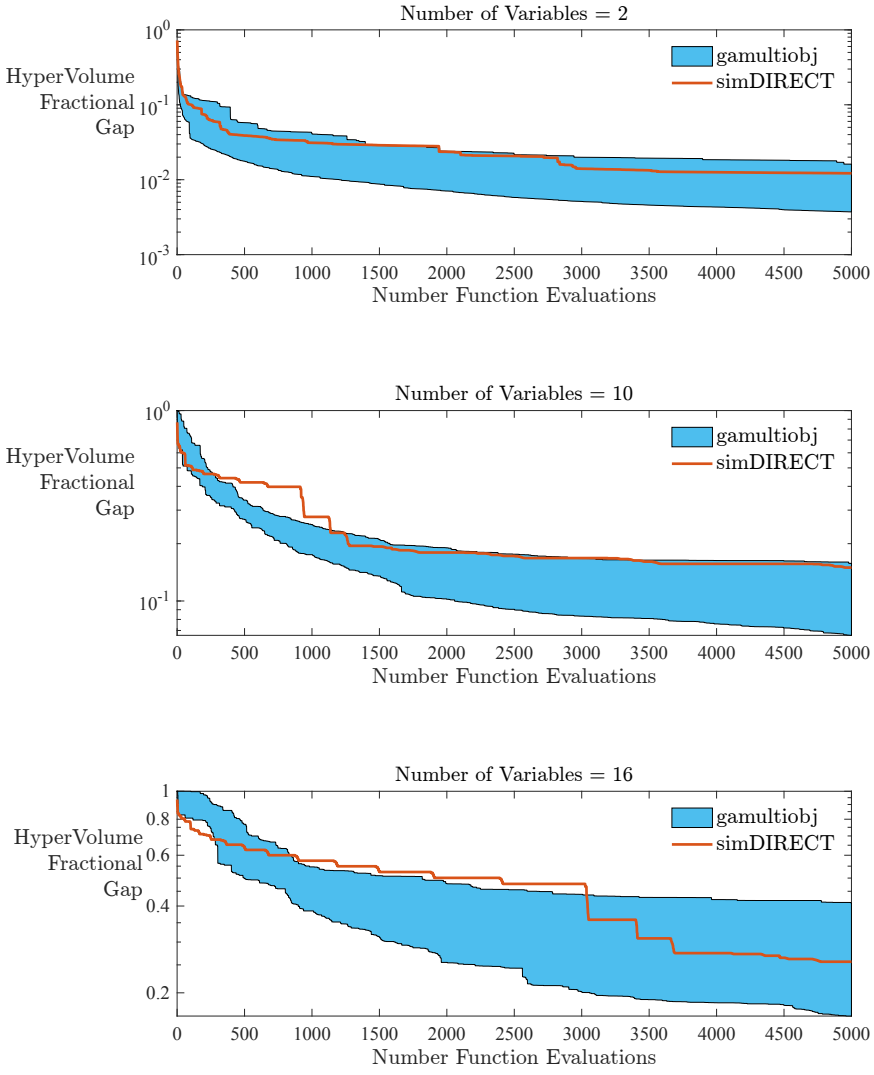

Figure 20: Comparing simDIRECT to gamultiobj in terms of progress in reducing the hypervolume fractional gap on the three-objective DTLZ2 problem with $x* = \sqrt{2}/2$.

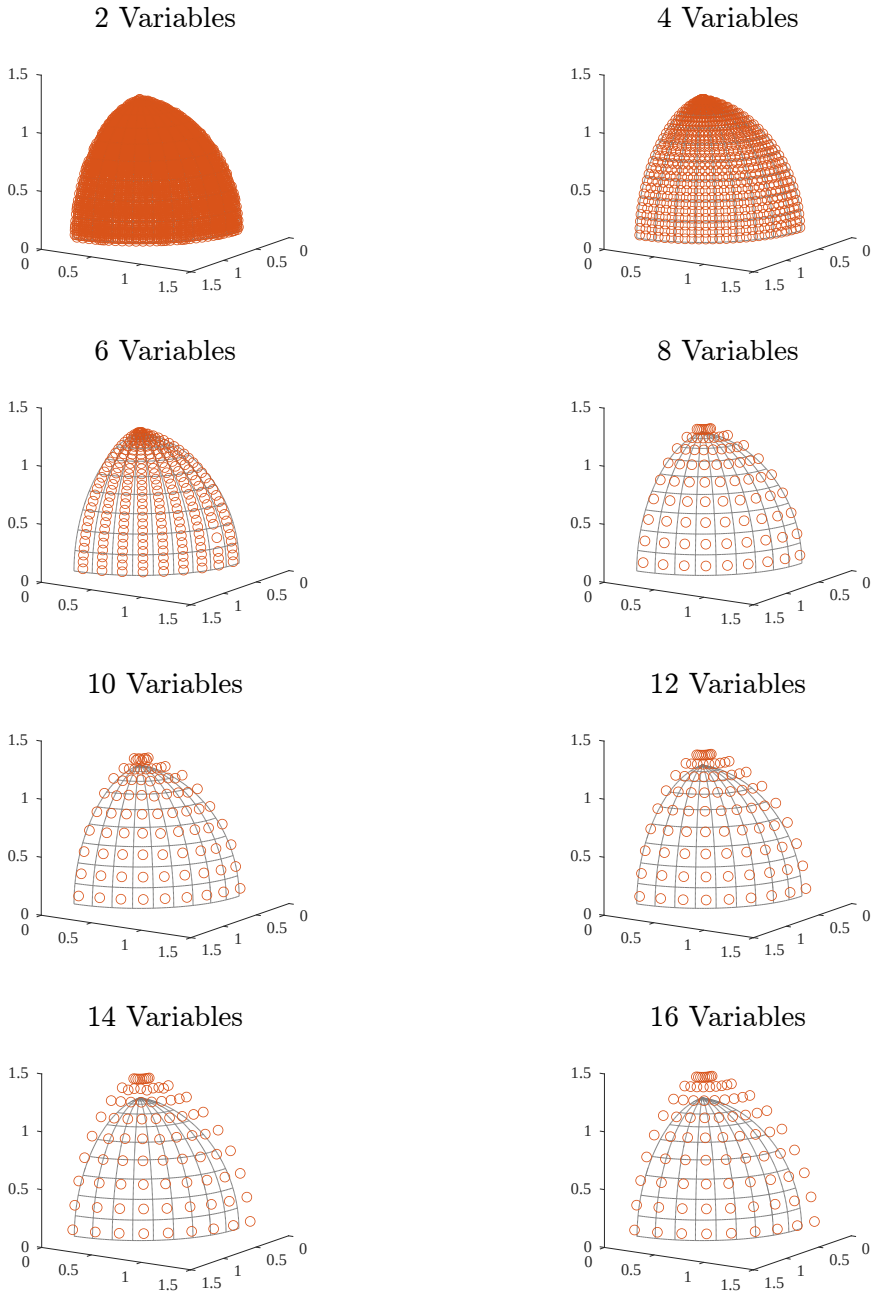

Figure 21: Nondominated sampled points from simDIRECT when run for 5000 function evaluations on the three-objective version of the DTLZ2 problem with $x* = \sqrt{2}/2$.

7 Adding constraints to simDIRECT

Most real-world multiobjective optimization problems have constraints. One type of constraint is a limit on how bad an objective can be. For example, in an automotive application, we may have two objectives: crash rating (1 to 5 stars, higher is better) and mass (lower is better). While we want to explore the tradeoff between crash rating and mass, there may be a limit on how low a crash rating we are willing to accept; for example, we might only want to consider designs that are three-star or higher. In this case, we would want to constrain the optimization so that it doesn't waste evaluations producing nondominated points with very low mass (very desirable) but only a one-star crash rating (unacceptable).

Besides bounds on the objectives, there may be side constraints involving other metrics that are more of a pass-fail nature; that is, once the metric meets the requirement (perhaps with some safety margin), there is no benefit to improving the metric further. For example, in order for a car to not overheat, the under-hood layout must provide sufficient air flow to cool critical components. Determining whether or not these air-flow requirements are satisfied may require running an aero-thermal simulation that predicts component temperatures, each of which must be below a certain limit. In this case, we would only want to meet the constraint with some safety margin, but cooling the component further would have no benefit.

In reality, the categorization of metrics as either objectives or constraints is somewhat arbitrary and often more a matter of emphasis; the more important metrics are made objectives so we can explore tradeoffs, and the less important metrics are considered constraints that just have to meet some threshold.

In this section we will show how to add constraints to simDIRECT; in particular, we will consider the problem

$$\begin{aligned}
&\underset{x_1,\ldots,x_d}{\text{minimize}} \quad (f_1(x_1,\ldots,x_d),\ldots,f_M(x_1,\ldots,x_d)) \\
&\text{subject to:} \\
&\qquad f_m(x_1,\ldots,x_d) \le f_m^U, && m = 1,\ldots,M \qquad (13)\\
&\qquad g_s(x_1,\ldots,x_d) \le 0, && s = 1,\ldots,S \qquad (14)\\
&\qquad x_k^L \le x_k \le x_k^U, && k = 1,\ldots,d. \qquad (15)
\end{aligned}$$

The above problem formulation contains the two kinds of constraints just discussed. In equation (13), we put limits on how bad an objective can

be, and in equation 14 we add side constraints. Equality constraints of the form $h(x_1, ..., x_d) = 0$ can be approximated by adding the two inequality constraints $h(x_1, ..., x_d) - \tau \leq 0$ and $-h(x_1, ..., x_d) - \tau \leq 0$ for some equality tolerance $\tau > 0$.

With this expanded problem formulation, we need to modify the concept of a "potentially Pareto optimal rectangle" to take into account constraints, producing a definition of a "potentially *feasible* and Pareto optimal rectangle." Several modifications to our previous Definition 2 are required.

First, in addition to considering Lipschitz constants K_m, $m = 1, ..., M$ for the objective functions, we will also need to consider Lipschitz constants L_s, $s = 1, ..., S$ for the side constraints in equation (14). Each rectangle will then have a vector of lower bounds for the objective functions, *and* a vector of lower bounds for the side constraints. In order for rectangle i to be "potentially *feasible* and Pareto optimal" we will require that these lower bounds be *feasible*, that is, equations (13) and (14) must be met if we replace the left hand sides of those equations with the lower bounds.

Second, we will only say "rectangle i dominates rectangle r" if the vector of objective lower bounds for rectangle i dominates that of rectangle r *and* the lower bounds for rectangle i's objectives and constraints are feasible. That is, in order for any rectangle to dominate another, it must first be feasible.

Putting all of these modifications together, we can now formally define the notion of a "potentially *feasible* and Pareto optimal rectangle":

Definition 3. Potentially feasible and Pareto optimal hyperrectangle
Let M be the number of objectives; S be the number side constraints; P be the set of feasible rectangles whose centerpoint outputs are not dominated by any other feasible rectangle; $\epsilon_m > 0$ be the desired accuracy for objective m; and f_m^U be an upper bound on how bad objective m can be ($m = 1, ..., M$). We say that rectangle i is *potentially feasible and Pareto optimal* if there exist positive constants $K_1, ..., K_M$ for the objectives and constants $L_1, ..., L_S$ for the side constraints such that the following conditions for rectangle i's lower bounds are met. First, the lower bounds for the objectives and side constraints are feasible:

$$\begin{aligned} f_m(\mathbf{c}_i) - K_m d_i &\leq f_m^U, && \text{for all } m = 1, ..., M \\ g_s(\mathbf{c}_i) - L_s d_i &\leq 0, && \text{for all } s = 1, ..., S \end{aligned}$$

Second, rectangle i's lower bounds are nondominated:

$$\begin{gathered}(f_1(\mathbf{c}_i) - K_1 d_i,\ \ f_2(\mathbf{c}_i) - K_2 d_i,\ \ldots,\ f_M(\mathbf{c}_i) - K_M d_i) \\ \text{is not dominated by} \\ (f_1(\mathbf{c}_j) - K_1 d_j,\ \ f_2(\mathbf{c}_j) - K_2 d_j,\ \ldots,\ f_M(\mathbf{c}_j) - K_M d_j)\,, \\ \text{for all } j \neq i \text{ that also satisfy} \\ f_m(\mathbf{c}_j) - K_m d_j \leq f_m^U, \quad \text{for all } m = 1, ..., M \\ g_s(\mathbf{c}_j) - L_s d_j \leq 0, \quad \text{for all } s = 1, ..., S\end{gathered}$$

Third, rectangle i's lower bounds improve the Pareto front non-trivially:

$$\begin{gathered}(f_1(\mathbf{c}_i) - K_1 d_i,\ \ f_2(\mathbf{c}_i) - K_2 d_i,\ \ldots,\ f_M(\mathbf{c}_i) - K_M d_i) \\ \text{is not dominated by} \\ (f_1(\mathbf{c}_p) - \epsilon_1,\ \ f_2(\mathbf{c}_p) - \epsilon_2,\ \ldots,\ f_M(\mathbf{c}_p) - \epsilon_M)\,, \\ \text{for all } p \in P\end{gathered}$$ □

In simDIRECT, we will select any rectangle that is potentially feasible and Pareto optimal as in Definition 3 using values of K_m and L_s that are proportional to the observed rates of change of the corresponding functions. If we let R_m^f and R_s^g denote the average absolute rate of change of objective m and side constraint s, respectively, then simDIRECT will select rectangle i if there exists an $\alpha > 0$ such that rectangle i meets the conditions of Definition 3 using $K_m = \alpha R_m^f$ and $L_s = \alpha R_s^g$.

The algorithmic implementation of this constrained selection rule is very similar to what we did in the unconstrained case (Algorithm 1). As before, we will decide whether or not to select rectangle i by starting with the set of all possible α values and then subtracting intervals $[a, b]$ of alpha values for which the conditions of Definition 3 are violated. If any α values remain after we make all possible subtractions, then rectangle i will be selected. The key differences from the unconstrained case are: (i) we must subtract any α values for which rectangle i is not feasible, and (ii), for some rectangle j to dominate rectangle i using some α, rectangle j's lower bounds must also be feasible with that α.

Now for rectangle i's lower bounds to be feasible, we require

$$f_m(\mathbf{c}_i) - \alpha R_m^f d_i \leq f_m^U, \qquad \text{for all } m = 1, ..., M \tag{16}$$

$$g_s(\mathbf{c}_i) - \alpha R_s^g d_i \leq 0, \qquad \text{for all } s = 1, ..., S \tag{17}$$

For equation (16) to hold for all $m = 1, ..., M$, we require

$$\alpha \geq \max_m \frac{f_m(\mathbf{c}_i) - f_m^U}{R_m^f d_i} \equiv a_i^f \tag{18}$$

Similarly, for equation(17) to hold for all $s = 1, ..., S$, we require

$$\alpha \geq \max_m \frac{g_s(\mathbf{c}_i)}{R_s^g d_i} \equiv a_i^g \tag{19}$$

Thus, to guarantee that rectangle i is feasible, we require

$$\alpha \geq a_i^{\min} \equiv \max\{a_i^f, a_i^g, 0\} \tag{20}$$

The algorithm for how simDIRECT selects rectangles in our constrained problem formulation is essentially the same as our previous Algorithm 1, except that we add equation (20) at the appropriate points – either to ensure that α is high enough to make rectangle i feasible, or to ensure that it is high enough that some other rectangle j not only dominates on the objectives, but also is feasible. The detailed pseudocode for "simDIRECT Rectangle Selection with Constraints" is given in Algorithm 3.

Algorithm 3 simDIRECT Rectangle Selection with Constraints

Input: Number rectangles n; number objectives M; objective values at rectangle centerpoints $f_m(\mathbf{c}_i)$; side constraint values at rectangle centerpoints $g_s(\mathbf{c}_i)$; rectangle center-vertex distances d_i; average absolute rates of change of objectives R_m^f and side constraints R_s^g; desired accuracy of objectives ϵ_m; upper bounds on the objectives f_m^U; set P of feasible rectangles whose centerpoint objectives are nondominated by the objectives of any other feasible rectangle.

Output: Set of selected rectangles.

Initialize the set of selected rectangles to be the empty set
For all $i = 1, ..., n$, compute $a_i^{\min}$ using equations (18)-(20)
for $i = 1, ..., n$ **do**
  Set $\mathcal{S} = \{[a_i^{\min}, \infty]\}$
  for $j = 1, ..., n$ and $j \neq i$ **do**
    if $d_j = d_i$ **then**
      if $f_m(\mathbf{c}_j) \leq f_m(\mathbf{c}_i)\ \forall\ m$, with at least 1 ineq. strict **then**
        Subtract $[a_j^{\min}, \infty]$ from $\mathcal{S}$
      end if
    else if $d_j > d_i$ **then**
      Compute $a_{ij} = \max_m \frac{f_m(\mathbf{c}_j) - f_m(\mathbf{c}_i)}{R_m(d_j - d_i)}$
      Compute $\tilde{a}_{ij} = \max\{a_{ij}, a_j^{\min}\}$
      Subtract $(\tilde{a}_{ij}, \infty]$ from $\mathcal{S}$
    else
      if $b_{ij} \equiv \min_m \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_j)}{R_m(d_i - d_j)} > a_j^{\min}$ **then**
        Subtract $[a_j^{\min}, b_{ij})$ from $\mathcal{S}$
      end if
    end if
  end for
  for each p in P **do**
    if $b_{ip} \equiv \min_m \frac{f_m(\mathbf{c}_i) - f_m(\mathbf{c}_p) + \epsilon_m}{R_m d_i} > 0$ **then** subtract $[0, b_{ip})$ from S
  end for
  if $\mathcal{S} \neq \emptyset$ **then** add rectangle i to the set of selected rectangles
end for

8 Numerical results with constraints

In this section we explore the performance of simDIRECT using two small constrained problems. Obviously, such a small test set is not enough to gauge the performance of any algorithm relative to the available methods in the literature. Our goal here is only to report our initial experience, to verify that simDIRECT can at least solve these simple problems, and to convince the reader that simDIRECT merits further testing and study.

8.1 The SRN problem

The first test problem is called SRN and is taken from Deb [18]. The problem has two variables, two objectives, and two constraints; we will use it to get an initial assessment of how simDIRECT performs relative to Matlab's "gamultiobj" when doing constrained, multiobjective optimization.

SRN Problem:	
$\underset{x_1,x_2}{\text{minimize}}$	$(\ f_1(x_1,x_2),\ f_2(x_1,x_2)\ )$
subject to	
	$g_1(x_1,x_2) = x_1^2 + x_2^2 - 225 \leq 0$
	$g_2(x_1,x_2) = x_1 - 3x_2 + 10 \leq 0$
	$-20 \leq x_1 \leq 20$
	$-20 \leq x_2 \leq 20$
where	
	$f_1(x_1,x_2) = 2 + (x_1-2)^2 + (x_2-1)^2$
	$f_2(x_1,x_2) = 9x_1 - (x_2-1)^2$
Nadir point:	
	$[1000, 100]$
Hypervolume:	
	$2.929719661183e+05$

Figure 22 shows the Pareto front for the SRN problem in both the output and input space. Both simDIRECT and Matlab's gamultiobj were used to solve this problem using a function-evaluation limit of 5000 (in simDIRECT we set $\epsilon_1 = \epsilon_2 = 0.01$). Figures 23 and 24 show the resulting sampled points in both the output and input spaces, respectively. Dominated points are shown using blue open circles and nondominated points are shown using red filled circles. While both methods obtain a good approximation to the Pareto

front, simDIRECT obtains a noticeably sharper approximation, especially in the input space. The relative advantage of simDIRECT also is visible in Figure 25 which compares the progress of the two methods in reducing the hypervolume fractional gap.

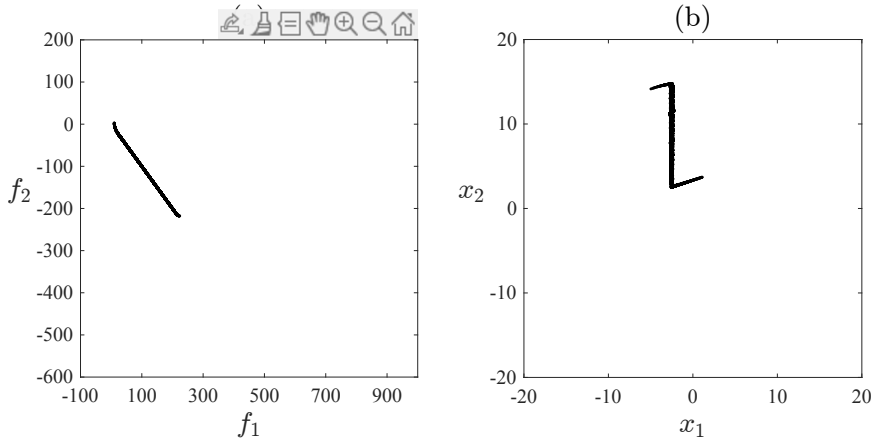

Figure 22: The Pareto front for the SRN problem: (a) output space; (b) input space.

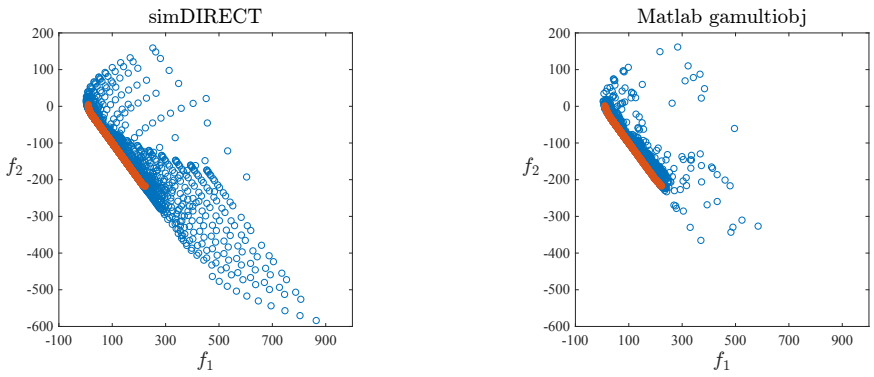

Figure 23: Sampled points from simDIRECT and Matlab's gamultiobj on the SRN problem, here in the output space. Open blue circles are dominated points and filled red circles are nondominated points.

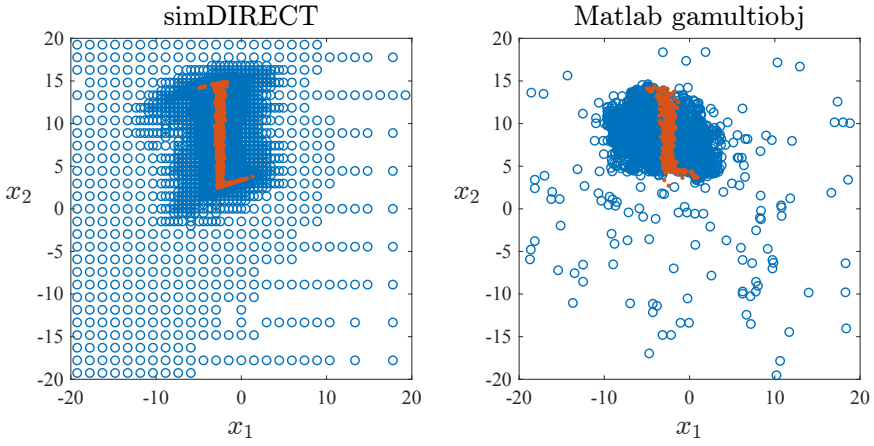

Figure 24: Sampled points from simDIRECT and Matlab's gamultiobj on the SRN problem, here in the input space. Open blue circles are dominated points and filled red circles are nondominated points.

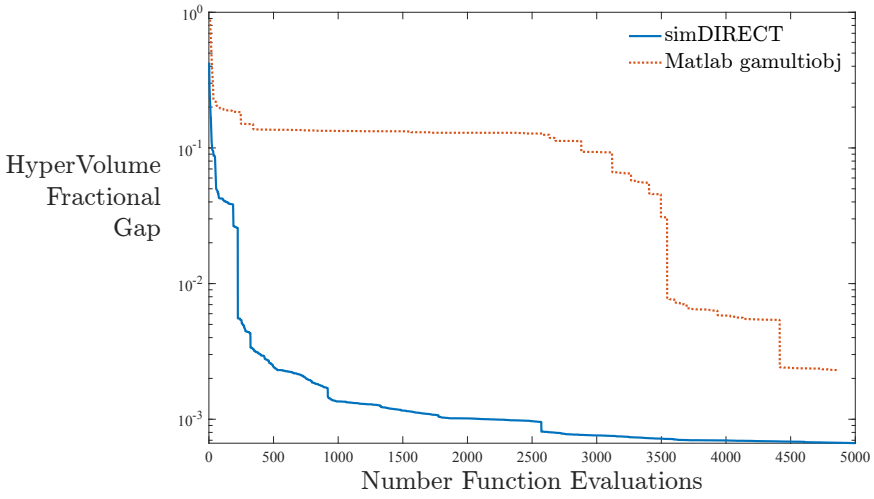

Figure 25: The convergence of the hypervolume fractional gap on the SRN problem for simDIRECT and Matlab's gamultiobj.

8.2 The Gomez #3 problem

The second problem is called the "Gomez #3" problem since it was listed as the third test problem in an article by S. Gomez and A. Levy [19]. This problem has two variables, one objective, and one constraint; we will use it to get an initial assessment of how simDIRECT performs on standard *single*-objective, constrained global optimization problems.

Gomez #3 Problem:

$$\begin{aligned}
&\underset{x_1,x_2}{\text{minimize}} && f(x_1,x_2) = \left(4 - 2.1x_1^2 + \tfrac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \\
&\text{subject to:} \\
& && g(x_1,x_2) = -\sin(4\pi x_1) + 2\sin^2(2\pi x_2) \le 0 \\
& && -1 \le x_1 \le +1 \\
& && -1 \le x_2 \le +1 \\
&\text{Optimum:} \\
& && x_1 = 0.10943 \\
& && x_2 = -0.62348 \\
& && f = -0.97110
\end{aligned}$$

What makes the Gomez #3 problem difficult is that the feasible region consists of several disconnected, roughly circular areas (shaded blue in Figure 26). The contour lines in Figure 26 are for the objective function, which clearly has two local minima. The global optimum (black dot in the figure) lies close to one of these local minima, on the boundary of one of the disconnected feasible subregions.

On this problem, simDIRECT quickly gets to within 1% of the global optimum in 145 function evaluations. (This run used desired accuracy of $\epsilon = 10^{-6}$, but the results are fairly insensitive to epsilon, giving exactly the same result for values between 10^{-2} and 10^{-8}). Figure 27 shows the sampled points and rectangles after 145 evaluations; for reference, the location of the global minimum is indicated by the red circle.

Figure 28 compares the convergence of simDIRECT to the convergence of my earlier (2001) extension of DIRECT to handle constraints [5] The 2001 version also calculated the average rate of change of the objectives and constraints and used these rates during rectangle selection (albeit in a different way). Being so related, one wonders which one tends to work best

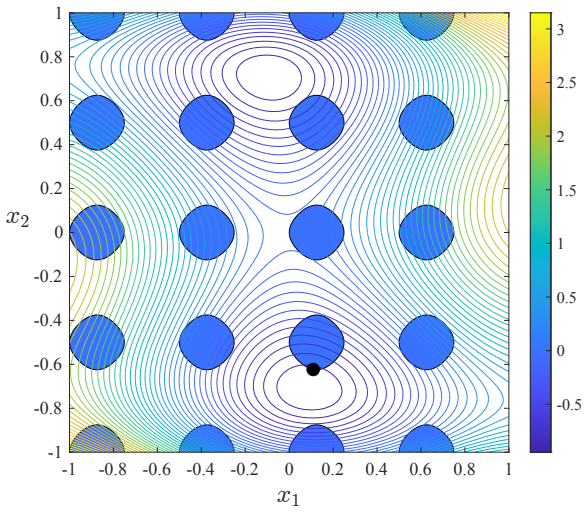

Figure 26: The Gomez 3 Problem. The feasible region consists of several disconnected, roughly circular areas (shaded blue). The contour lines are for the objective function.The black dot shows the location of the constrained global minimum.

on constrained *single*-objective problems. For the Gomez #3 problem, the convergence rates shown in Figure 28 are virtually equivalent. On other problems (not shown here), sometimes simDIRECT is better and sometimes worse. Conceptually, however, simDIRECT seems more appealing, as it stays much truer to the original Lipschitzian motivation of DIRECT. A more thorough comparison of the two approaches is left to future research.

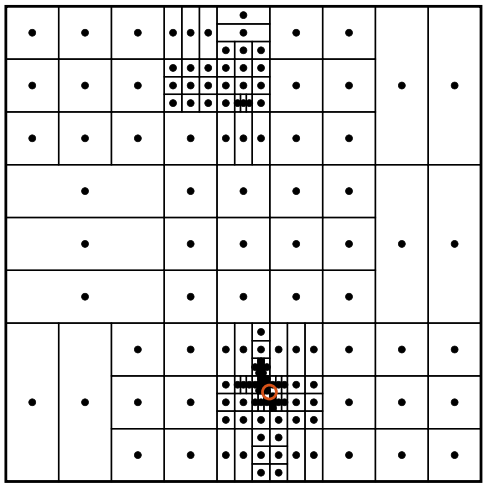

Figure 27: Status of simDIRECT on the Gomez 3 Problem after 145 function evaluations. The location of the global minimum is indicated by the red circle.

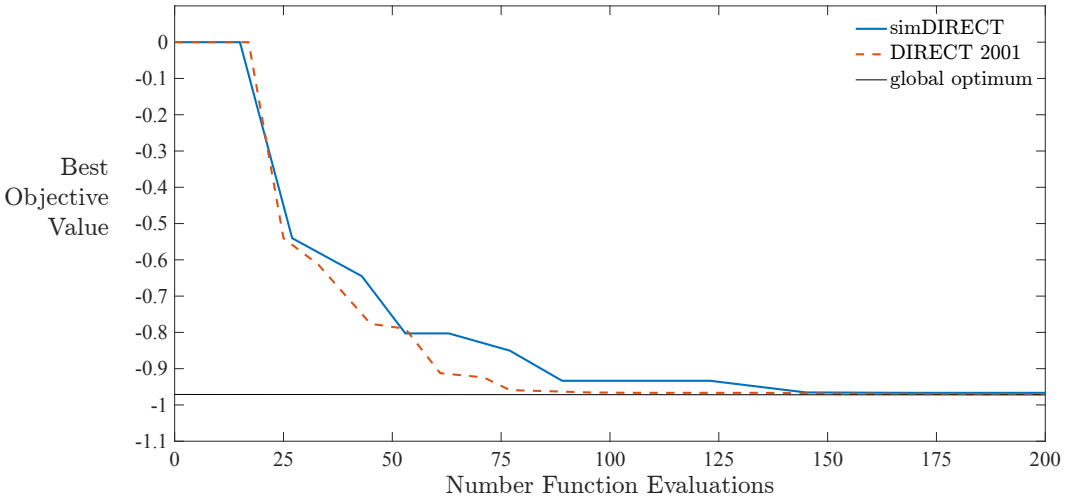

Figure 28: Iteration history on the Gomez #3 problem of simDIRECT and the constrained DIRECT algorithm of [5]

9 Handling failed runs

When applying optimization to complex computer simulations, it may happen that the simulation occasionally fails to provide results for certain input parameters. In some cases, the failure may be due to computational reasons (segmentation error, negative volume elements, mesh penetration, etc.). In other cases, the "failure" is that the objective or constraint function is not defined. For example, suppose we are optimizing the control parameters of a robot in order to minimize the time it takes to complete a task. In this case, there may be parameter settings where the robot fails to complete the task (e.g., it is too aggressive and hits an obstacle), and so the "time to complete the task" is undefined.

Failed runs are not that serious a concern if the failure happens far from the optimum. In such cases one can usually identify regions where failure occurs and prevent the optimization algorithm from ever visiting these regions by imposing bounds on the parameters or adding simple constraints. But when failures occur near the global optimum, attempts to avoid failures via bounds or adding simple constraints run the risk of inadvertently excluding the optimum. Moreover, the simple scheme of using a very large objective value for failed runs will not work, as it would cause simDIRECT to take a very large number of iterations before selecting any rectangle whose centerpoint simulation failed – effectively excluding the optimum if it lies in such a rectangle.

With simDIRECT, there is a simple and effective way to handle failed runs. We have to do three things. First, of course, we keep track of the set of failed runs, which we will call $\mathcal{F}$. Second, before beginning the process of rectangle selection, for each failed run, we replace the missing output values with the values from the closest point where the run did *not* fail (we compute distances using the data normalized to $[0, 1]$). Filling in missing data based on the nearest sucessful run is not a new idea, as it appeared as early as 2001 in J. Gablonsky's Ph.D. thesis [6].

What is new is the third step. We create an additional constraint function $g_{S+1}(\cdot)$ whose value is the distance to the closest point where the simulation does *not* fail. We then require $g_{S+1}(\mathbf{x}) \leq 0$, that is, we require the simulation to not fail. For each point $i \in \mathcal{F}$, we set $g_{S+1}(\mathbf{c}_i)$ equal to the distance to the closest non-failed point that we have sampled so far, and for all $i \notin \mathcal{F}$ (i.e., successful runs) we set $g_{S+1}(\mathbf{c}_i) = 0$. Finally, we set the average rate of change of this new constraint to 1, for the following reason. Consider

any $i \in \mathcal{F}$ and suppose that rectangle j is the nearest non-failed point with distance $\|\mathbf{c}_i - \mathbf{c}_j\| = \delta_i$; that is, $g_{S+1}(\mathbf{c}_i) = \delta_i$. If we move in a straight line from $\mathbf{c}_i$ to $\mathbf{c}_j$, the change in $g_{S+1}(\cdot)$ will be $|g_{S+1}(\mathbf{c}_i) - g_{S+1}(\mathbf{c}_j)| = |\delta_i - 0| = \delta_i$. This change happens over a distance $\|\mathbf{c}_i - \mathbf{c}_j\| = \delta_i$. Hence, the rate of change is $\delta_i/\delta_i = 1$.

Having added this extra constraint, we then do selection as usual. However, there is one special case that must be considered, namely, the case when the all the runs made so far have failed. In this case, there will be no "closest non-failed point." What we do in this case is simple: lacking any information, we simply select all the rectangles.

Why does this work? Well, suppose point i is a failed run and point j is the nearest non-failed point. Since we fill in the missing data for point i by setting them equal to the values from point j, our "failed" rectangle i looks very similar to our "not failed" rectangle j. The only differences are: (i) the rectangles differ on the new constraint (failed rectangle worse) and, (ii), the rectangles may differ in size. As a result, if the failed rectangle's size is strictly less the size of the non-failed rectangle ($d_i < d_j$), then it will be dominated by the non-failed rectangle and not be selected. At the other extreme, if the failed rectangle is larger ($d_i > d_j$), then the lower bounds on the objective for the failed rectangle will be better than those for the non-failed. So the failed rectangle might even dominate the non-failed; in fact, it will do so for α large enough that the lower bounds for all the constraints are feasible.

Putting it all together, there will be a tendency for failed rectangles to have a similar size as the nearest non-failed rectangle. As a result, if the non-failed rectangle is selected and subdivided, it will be more likely that a *nearby* failed rectangle will be selected and subdivided.[2] In short, this way of handling failed runs naturally causes the algorithm to explore "failed" rectangles if they are near the Pareto front of "non-failed" rectangles.

To illustrate how this idea works, let's consider the Gomez #3 problem and imagine that the simulation fails for all points in the shaded red triangle in Figure 29.

Because the failure region passes very close to the optimum, this failure is of the more troublesome sort. Figure 30 compares the rate of convergence of

[2]For a failed rectangle i to be selected, it must be larger than the nearest non-failed rectangle. Moreover, α must be at least δ_i/d_i for the lower bound on the new constraint to be feasible, so the smaller the distance to the nearest feasible point (δ_i), the smaller will be the required α, and hence the more likely it is that the failed rectangle will be selected.

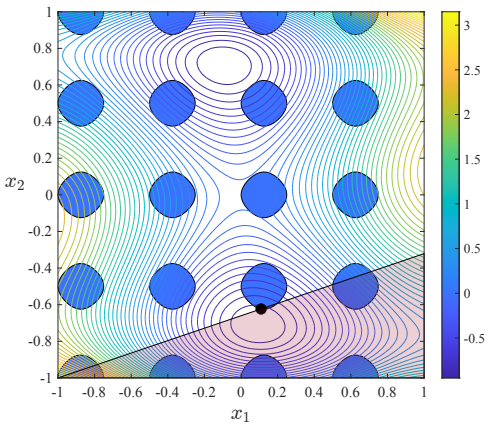

Figure 29: To illustrate how simDIRECT handles failed runs, we use the same Gomez #3 problem discussed earlier, but assume that the simulation fails (returns no output) for all points in the shaded red triangle above.

simDIRECT on the Gomez #3 problem with and without failures. Where it previously took 145 evaluations to get within 1% of the optimum, it now takes 195. Overall, the convergence rates are similar. Finally, Figure 31 shows the rectangles and sampled points after 195 evaluations on the problem with run failure. Comparing this to the case without run failure (Figure 27), we see a similar pattern.

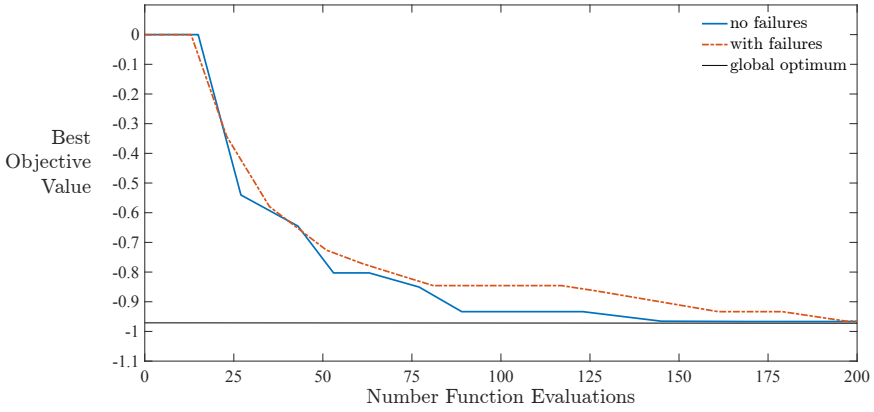

Figure 30: Convergence history of simDIRECT on the Gomez #3 problem with and without the region of failed runs shown in Figure 29.

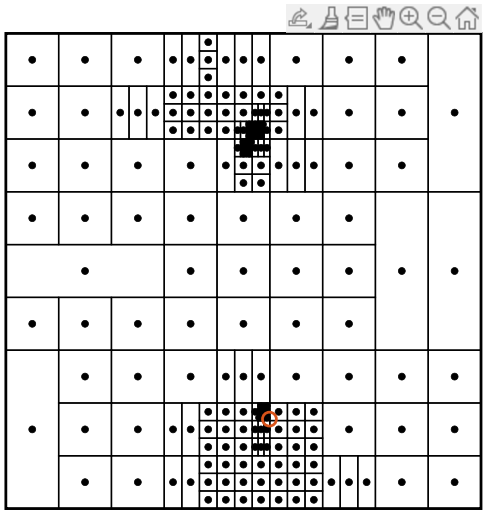

Figure 31: Sampled points and rectangles from running simDIRECT for 195 function evaluations on the Gomez #3 problem with the region of failed runs shown in Figure 29.

10 Software

A straightforward implementation of simDIRECT in the Matlab programming language is available on GitHub under a permissive MIT license (`https://github.com/donaldratnerjones/simDIRECT`). The code has been written for maximum readability, and should therefore help resolve any questions the reader may still have as to how simDIRECT works. Any readers who use the code are encouraged to send questions, comments, and feedback to `donjon@umich.edu`.

11 Discussion

The world of global optimization algorithms can be roughly divided into two branches distinguished by their source of inspiration. The most recent branch consists of "nature-inspired heuristics" such as simulated annealing, genetic algorithms, particle swarm optimization, firefly algorithms, cuckoo search, and bat algorithms [20]. These algorithms are stochastic; treat the objective and constraint functions as black boxes having no exploitable properties beyond continuity; and can only guarantee convergence to the global optimum in the limit as the number of function evaluations goes to infinity. The second, older branch consists of "mathematics-inspired algorithms" such as linear programming, branch-and-bound methods, cutting-plane methods, outer approximation methods, Lipschitzian methods, and interval analysis [21]. These methods are deterministic; make explicit assumptions about the properties of the objective and constraint functions; and often can prove convergence to the global optimum within some tolerance in a finite number of function evaluations.

The attraction and charm of DIRECT has always been the way it bridges the gap between these extremes. Inspired by Lipschitzian branch-and-bound methods, it is deterministic and mathematics-inspired; at the same time, it treats the problem functions as black-boxes and has the simplicity and intuitive appeal of the best nature-inspired methods.

Dampening this appeal is the fact that the original DIRECT algorithm [9] is limited in its potential applications: it assumed a single objective, only continuous (non-integer) variables, no constraints beyond bounds on the variables, and no failed runs. As a result, many papers were written that offered extensions of DIRECT to fill in these gaps (see [10] for a recent review). Un-

fortunately, many of these extensions are unsatisfying because, in the effort to add new capability to DIRECT, they also add high complexity as well as new algorithmic tuning parameters. With simDIRECT, I believe we now have found a way to add substantial capability to DIRECT in a way that stays true to its original Lipschitizian foundation, preserves its conceptual and algorithmic simplicity, and avoids adding new tuning parameters.

While we haven't detailed it here, the extension to handling integer variables described in [5] can also be added to simDIRECT, as it only requires modifying rectangle trisection to work with integer variables. With this extension, simDIRECT can claim a truly impressive catalog of capabilities: single or multiple objectives, black-box inequality and equality constraints, continuous or integer variables, the ability to handle failed runs, and deterministic search (no sensitivity to random number seed).

Of course, no algorithm can do everything well – there's no free lunch – and so, for the sake of balance, it worth also recalling DIRECT's limitations and weaknesses. First among the limitations is the need to have tight lower and upper bounds on the variables. In order to converge in a reasonable amount of time, DIRECT needs to be able to get close to the optimum after trisecting on each dimension only a few times (say 10-15); so if all one knows about a variable is that it is positive, using very weak bounds like $[0, 10^6]$ is not likely to work well, as it would take many trisections just to get in the vicinity of the optimum. The second critical limitation is on the number of variables. The results given in the original paper [9], as well as those shown here, suggest that DIRECT is very strong with 1-6 variables, reasonably strong with 7-12, and – with a large enough function evaluation budget – sometimes effective up to 16 variables. Of course, such general statements about problem size are only general tendencies; everything depends on the difficulty of the particular problem at hand. For example, Gaviano et al. [22] have produced a test-function generator for global optimization called "GKLS" that can produce problems with a specified degree of difficulty, so difficult that DIRECT takes over 100,000 evaluations on problems with only 3-5 variables [23].

While we are enthusiastic about simDIRECT, the book is certainly not closed with respect to new and useful extensions and/or improvements to the original DIRECT or the new simDIRECT. One especially fertile area for research is on ways to hybridize DIRECT with other approaches. For example, for multiobjective optimization, simDIRECT could be run for (say) 600 evaluations and the best 200 of these used as the initial population for

a multiobjective evolutionary algorithm. Similarly, Bayesian optimization methods that now use a Latin Hypercube to initialize the search could, instead, run DIRECT for several iterations to get an initial set of points.

With respect to simDIRECT itself, several research topics come to mind. First, in simDIRECT, we require the Lipschitz constants K_m in the definition of "potentially Pareto optimal rectangle" to be some multiple α of the average, absolute rates of change observed so far over the course of the search. This sounds reasonable, but then again: why use the *average* rate of change? Why not use the maximum, the median, or the 95% quantile? There is definitely room for a study that evaluates these alternatives on a good set of test functions and determines if one of them has a clear advantage.

Second, we have noticed that the number of points per iteration rises rapidly with each simDIRECT iteration on problems with 2 or 3 objectives, sometimes reaching over 1000 points per iteration when using a function-evaluation limit of 5000. This makes sense because, unlike single-objective optimization where we usually are converging to a point, with 2-3 objectives we are converging to the *Pareto front*, which is a 1-2 dimensional manifold. Even so, with so many points being sampled per iteration, the function evaluation budget is soon exhausted. Thus we wonder if it may be useful to limit the number of selected rectangles by first running selection as usual and then, if we get more than (say) 200 selected rectangles, somehow selecting the "best" 200 of them for trisection.

Finally, we have left one obvious question begging for an answer: how does simDIRECT's performance compare to that of the *multi*DIRECT algorithm that was its inspiration? Going forward, I hope to interest the inventors of *multi*DIRECT, Alberto Lovison and Kaisa Miettinen, in running their implementation of *multi*DIRECT on the test problems used here, so we may get some head-to-head comparisons. Until then, we only know that *multi*DIRECT always selects more rectangles than simDIRECT, and so executes a wider search. What we don't know is whether this extra sampling helps find the optimum sooner, or if it just adds overhead and slows down the search. It will be interesting find out.

Aknowledgments

I would like to thank Joaquim R. R. A. Martins for helpful feedback on an early draft as well as assistance in debugging the LaTeX formatting.

References

[1] A. Lovison and K. Miettinen, "On the extension of the direct algorithm to multiple objectives," *Journal of Global Optimization*, vol. 79, no. 2, pp. 387–412, 2021.

[2] K. Deb, *Multi-Objective Evolutionary Algorithms*, pp. 995–1015. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[3] L. M. Pang, H. Ishibuchi, and K. Shang, "Offline automatic parameter tuning of moea/d using genetic algorithm," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1889–1897, 2019.

[4] S. Akhmedova and V. Stanovov, "Success-history based parameter adaptation in moea/d algorithm," in *Advances in Swarm Intelligence* (Y. Tan, Y. Shi, and M. Tuba, eds.), (Cham), pp. 455–462, Springer International Publishing, 2020.

[5] D. R. Jones, "Direct global optimization algorithm," in *Encyclopedia of Optimization* (C. A. Floudas and P. M. Pardalos, eds.), pp. 431–440, Boston, MA: Springer US, 2001. `https://www.researchgate.net/publication/249598200_The_DIRECT_global_optimization_algorithm#fullTextFileContent`.

[6] J. Gablonsky, *Modifications of the DIRECT Algorithm*. PhD thesis, North Carolina State University, Raleigh, NC, USA, 2001. `https://repository.lib.ncsu.edu/handle/1840.16/3920`.

[7] J. Na, Y. Lim, and C. Han, "A modified DIRECT algorithm for hidden constraints in an LNG process optimization," *Energy*, vol. 126, pp. 488 – 500, 2017.

[8] L. Stripinis and R. Paulavičius, "A new direct-glh algorithm for global optimization with hidden constraints," *Optimization Letters*, 2021.

[9] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *Journal of Optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, 1993.

[10] D. R. Jones and J. R. R. A. Martins, "The DIRECT algorithm: 25 years later," *Journal of Global Optimization*, vol. 79, no. 3, pp. 521–566, 2021.

[11] J. Mockus, "On the pareto optimality in the context of Lipschitzian optimization," *Informatica*, vol. 22, no. 4, pp. 521 – 536, 2011.

[12] C. S. Y. Wong, A. Al-Dujaili, and S. Sundaram, "Hypervolume-based DIRECT for multi-objective optimisation," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, (New York, NY, USA), pp. 1201–1208, ACM, 2016.

[13] C. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 1157–1163, 2006.

[14] A. Al-Dujaili and S. Suresh, "Dividing rectangles attack multi-objective optimization," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3606–3613, 2016.

[15] M. E. Hartikainen and A. Lovison, "Paint–sicon: constructing consistent parametric representations of pareto sets in nonconvex multiobjective optimization," *Journal of Global Optimization*, vol. 62, no. 2, pp. 243–261, 2015.

[16] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente, "Direct multisearch for multiobjective optimization," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 1109–1140, 2011.

[17] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 1, pp. 825–830 vol.1, 2002.

[18] K. Deb, P. C. Roy, and R. Hussein, "Surrogate modeling approaches for multiobjective optimization: Methods, taxonomy, and results," *Mathematical and Computational Applications*, vol. 26, no. 1, 2021.

[19] S. Gomez and A. V. Levy, "The tunnelling method for solving the constrained global optimization problem with several non-connected feasible regions," in *Numerical Analysis* (J. P. Hennart, ed.), (Berlin, Heidelberg), pp. 34–47, Springer Berlin Heidelberg, 1982.

[20] X.-S. Yang, *Nature-Inspired Optimization Algorithms*. Elsevier, 2014.

[21] C. A. Floudas, *Deterministic Global Optimization*. Springer US, 2000.

[22] M. Gaviano, D. E. Kvasov, D. Lera, and Y. D. Sergeyev, "Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization," *ACM Trans. Math. Softw.*, vol. 29, pp. 469–480, Dec. 2003.

[23] R. Paulavičius, Y. D. Sergeyev, D. E. Kvasov, and J. Žilinskas, "Globally-biased BIRECT algorithm with local accelerators for expensive global optimization," *Expert Systems with Applications*, vol. 144, p. 113052, 2020.