SANDIA REPORT

SAND2008-5844
Unlimited Release
Printed September 2008

Concurrent Optimization with DUET: DIRECT Using External Trial Points

Noam Goldberg, Tamara G. Kolda, Ann S. Yoshimura

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2008-5844
Unlimited Release
Printed September 2008

Concurrent Optimization with DUET: DIRECT Using External Trial Points

Noam Goldberg
Rutgers Center for Operations Research (RUTCOR)
Rutgers University
New Brunswick, NJ 08854
ngoldberg@rutcor.rutgers.edu

Tamara G. Kolda and Ann S. Yoshimura
Sandia National Laboratories
Livermore, CA 94551-0969
tgkolda,asyosh@sandia.gov

Abstract

We consider the general problem of minimizing a real function subject to bound constraints. We are motivated by applications in which the derivatives of the objective function are unavailable or expensive to compute. This can occur, for example, when the function is not given analytically and is evaluated by running a simulation. The DIRECT algorithm by Jones, Pertunnen, and Stuckman is a global derivative-free optimization algorithm. It is a deterministic algorithm that samples the space following a scheme that is justifiable theoretically in case of Lipschitz continuous functions. In practice, DIRECT has proven to be efficient, with respect to the number of function evaluations, in finding a global minimum for a wide variety of functions. We extend the DIRECT algorithm to make use of additional free trial points that are made available when running DIRECT simultaneously with other optimization algorithms. We compare several deterministic and randomized variants of the DIRECT algorithm that use the external trial points (DUET). We present experimental results with external trial points that are sampled at random to show an improvement of DUET over the performance of the DIRECT algorithm.

This page intentionally left blank.

Contents

Figures

Tables

This page intentionally left blank.

1 Introduction

It is of great interest in applications in science and engineering to be able minimize general non-convex real functions. DIRECT, by Jones, Pertunnen, and Stuckman [12], is a derivative-free global optimization algorithm for minimizing real functions subject to simple bound (or box) constraints. DIRECT employs a deterministic sampling scheme which has been shown to cluster around global optima. Moreover DIRECT is guaranteed to eventually converge to a global optimum [12, 4]. The sampling scheme involves the iterative refinement of the feasible space (a hyperrectangle) into smaller hyperrectangles. At each iteration a subset of the hyperrectangles that partition the space are selected to be further refined. The selection of hyperrectangles is done according to a rule that is theoretically justified for Lipschitz continuous functions.

Lipschitz optimization aims to find a global optimum while using the Lipschitz constant or an estimate [10]. Lipschitzian optimization exploits the bound that can be computed over a subset of the feasible region based on a given function value and the Lipschitz constant. The DIRECT algorithm can be viewed as Lipschitzian optimization that tries all possible values of the Lipschitz constant. This is accomplished by a selection rule that corresponds to selecting hyperrectangles that lie on the lower convex hull of a plot of rectangle size versus rectangle function value. A user defined parameter is used to balance the tradeoff of considering unexplored large rectangles (global search) versus rectangles that contain lowest function values (local search).

Direct search algorithms can be viewed as sampling methods in which function values are evaluated at a mesh that is iteratively refined. We would like the points sampled to cluster around global optima. When running optimization algorithms in parallel, hybrid optimization involves the use of information made available by other algorithms. We consider extensions to DIRECT as a sampling method so that it may take advantage of external trial points that are freely available.

In Section 2, we start with a description of the bound constrained global minimization problem and give an overview of the DIRECT algorithm by Jones et al. In Section 3, we describe our problem of extending DIRECT to use external trial points that are made available when running other optimization algorithms in parallel. We describe three different variants of the DUET (DIRECT Using External Trial-points) class of algorithms. We present an experimental study in Section 4, comparing the performance of the three different variants and the DIRECT algorithm by Jones et al. We present conclusions in Section 5.

This page intentionally left blank.

2 Problem description and background

We consider the global optimization problem of minimizing $f(x)$, where $f : \mathbb{R}^n \to \mathbb{R}$, subject to the bound constraints $l_i \leq x_i \leq u_i$ for $i = 1, ..., n$.

2.1 Background: Lipschitzian optimization and DIRECT

Lipschitzian optimization assumes $f(x)$ is Lipschitz continuous over the feasible region, with a known Lipschitz constant C so that for every $x, y \in \mathbb{R}^n$ such that $l_i \leq x_i, y_i \leq u_i$ for all $i = 1, \ldots, n$, the following holds:

$$|f(y) - f(x)| \leq C|y - x|.$$

The DIRECT algorithm by Jones et al. [12] does not require a priori knowledge of the Lipschitz constant. It successively partitions the search space (a hyperrectangle) into smaller hyperrectangles. At each iteration, a subset of *potentially optimal* hyperrectangles, see Definition 2.1, is chosen to be subdivided. Jones et al. compute a lower bound on the function value that can be obtained in each hyperrectangle using the function value of a hyperrectangle, denoted $f(R)$, and the distance, denoted $size(R)$, from the center point, denoted $center(R)$, to the vertices of R. More recent work has extended the definition of potentially optimal rectangles to use various measures of rectangle size, e.g., the L_∞ distance between a pair of vertices [8]. Jones et al. [12], use $f(R) = f(center(R))$, for the rectangle values $f(R)$. We generalize the rule to use function values other than at the center points.

Definition 2.1 *A hyperrectangle R is said to be* potentially optimal *if, for some constant $K > 0$ and all hyperrectangles R', it satisfies*

$$f(R) - K \cdot size(R) \leq f(center(R')) - K \cdot size(R'), \text{ and} \tag{2.1}$$
$$f(R) - K \cdot size(R) \leq f_{min} - \epsilon|f_{min}|, \tag{2.2}$$

where ϵ is a positive constant and f_{min} is the current best function value.

Finding *potentially optimal* rectangles can be done by solving a two dimensional convex hull problem. More specifically, we look for a lower convex hull in the function value versus hyperrectangle space space. Finding the lower convex hull in two dimensions can be done in linear time[1]. In the convex hull computation of function values there is an additional point, in the size, function value space, $(0, f_{min} - \epsilon|f_{min}|)$, which corresponds to equation (2.2). The epsilon value suggested by Jones et al. is 10^{-4}, which is the value that we use in our experimental results.

[1] Jones et al. [12] make use of Graham scan which is an $O(n \lg n)$ algorithm. We note that we can make use of a simple linear time algorithm for finding the lower convex hull if we maintain the hyperrectangles sorted by size.

2.2 An iterative version of DIRECT for parallel function evaluation

Algorithm 2.1 implements an iteration of the DIRECT algorithm which allows for parallel function evaluations. The algorithm is initialized with *partition* containing a single hypercube $R = [0,1]^n$, corresponding to the entire feasible region after normalizing each dimension to $[0,1]$, and having a known function value $f(R)$. The *partition* is updated each iteration by removing the *potentially optimal* hyperrectangles and replacing them with the smaller hyperrectangles into which they have been refined[2]. Algorithm 2.1 returns the next set of points to be evaluated, denoted by *newPoints*. It is then called with the set *evalPoints* containing the same points once they have been evaluated (in the first iteration this set would be initialized to be empty).

Algorithm 2.1 Parallel DIRECT
in: *evalPoints*
out: *newPoints*

ContainingRects $\leftarrow \emptyset$
for all $p \in$ *evalPoints* **do**
  Find rectangle $R \in$ *partition* s.t. $p \in R$
  Save p as a *subpoint* of R.
  ContainingRects $\leftarrow$ *ContainingRects* $\cup \{R\}$
end for
for all $R \in$ *ContainingRects* **do**
  Using *subpoints* order the long sides of R as $i_1, i_2, ..., i_m$ such that $w_{i_1} \leq w_{i_2} \leq ... \leq w_{i_m}$
  Rnew $\leftarrow$ *Divide*$(R, \{i_1, i_2, ..., i_m)\}$
  partition $\leftarrow$ (*partition* $\setminus \{R\}$) $\cup$ *Rnew*
end for
potOptimal $\leftarrow$ *FindPotOptimalRect*(*partition*)
for all $R \in$ *potOptimal* **do**
  newPoints $\leftarrow$ *newPoints* $\cup \{p \in R | p$ is a subpoint of $R\}$
end for

We define *subpoints* in order to describe the points that the DIRECT algorithm selects to be evaluated at each iteration.

Definition 2.2 *A point $p_i \in R \subset \mathbb{R}^n$ is called a* subpoint *of hyperrectangle R, along dimension i, if $p_i \in \{center(R) \pm \frac{\delta}{3} e_i\}$, where δ is the side length of R along dimension i.*

The DIRECT algorithm by Jones et al. [12] subdivides each long side length along dimension $i \in I$, where I is the set of dimensions corresponding to the long side length, following an

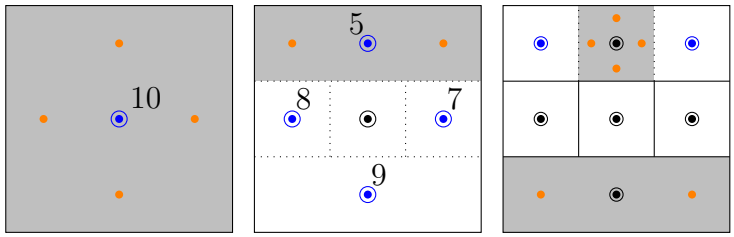
[2]In the implementation we actually store a fast ternary tree structure for the purpose of finding the Rectangle that contains a point. We also store the current partition sorted by size and then by function value. In order to save on insertions into the partitions we only insert a rectangle once it has a function value.

increasing order of:

$$w_i = \min\{f(p) : p \in \{center(R) \pm \frac{\delta}{3} e_i\}\}. \tag{2.3}$$

The Divide procedure for subdividing a hyperrectangle used by our DIRECT implementation generalizes the procedure of Jones et al. [12] and is described in the next section. When used in the above Algorithm 2.1, it is exactly the same as the Divide procedure in Jones' et al., unless we add additional trial points or change the order of the hyperrectangle sides (which increases in w_i).

In Figure 2.1 the first three nontrivial iterations of Algorithm 2.1 are shown. In Iteration 1, the four subpoints of the hyperrectangle corresponding to the entire feasible region are returned as *newPoints*. In Iteration 2, the subhyperrectangle to the right of the center is largest (along with the one on the left) and also has the minimum function value and therefore is the only potentially optimal hyperrectangle. The lower convex hull of hyperrectangles, in the function value versus hyperrectangle size space, that determines the set of potentially optimal rectangles in Iteration 3 is shown in Figure 2.2.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 2.1. Example of the (parallel) DIRECT algorithm. The first 3 nontrivial iterations of the algorithm are shown. *evalPoints* and *newPoints* are shown as blue and orange points, respectively. Black points represent existing points (that have been evaluated in previous iterations). The current subdivision is shown as a dotted line and potentially optimal rectangles are shaded.

In practice, the DIRECT algorithm is effectively used for optimization of functions that are not necessarily Lipschitz continuous. As a derivative-free optimization method, it can be effectively used for functions that are not given in analytical form. The function values could be expensive to compute, for example, the function may be the output of a running of a simulation.

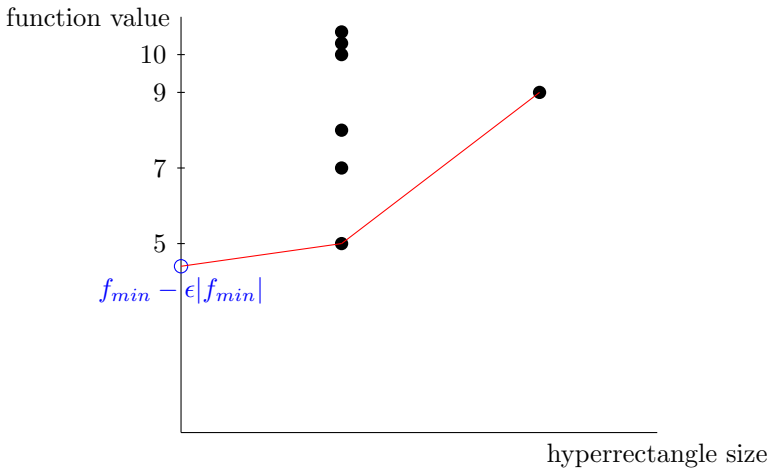


Figure 2.2. An example of a convex hull computation for selecting potentially optimal rectangles in Iteration 3 of Figure 2.1.

3 Extending DIRECT for hybrid optimization

When optimizing in parallel, the DIRECT algorithm may be run concurrently with a variety of other local and global optimization algorithms. When function calls are expensive, we would like the DIRECT algorithm to make use of additional free function evaluations that are made available by the other algorithms that are running simultaneously. In other words, we would like to extend the DIRECT algorithm to make use of external trial points (i.e., additional function evaluations from other optimization methods). We call the resulting algorithm DUET, short for DIRECT Using External Trial points. Our purpose it to investigate the effect of using the free function evaluations on the convergence of the algorithm to a global optimum.

Before we proceed, we need to modify our definition of $f(R)$. As you recall, we previously specified that $f(R) = f(center(R))$, but now we need to account for the additional points. We denote the set of all evaluated points in hyperrectangle R as $knownPoints(R)$. We will make use of the following generalized definition of $f(R)$:

$$f(R) = \begin{cases} +\infty & \text{if } knownPoints(R) = \emptyset, \\ \min\{f(p) : p \in knownPoints(R)\} & \text{otherwise.} \end{cases} \tag{3.1}$$

3.1 Dividing rectangles and splitting orders

Before we discuss the algorithms in detail, we present a version of the rectangle division procedure that has been modified to accommodate the extra points. DIRECT refines (i.e., subdivides) each potentially optimal rectangle into smaller rectangles. We generalize Jones' *Divide* algorithm [12], to subdivide hyperrectangles following a given order of the long side lengths of the hyperrectangle. The resulting algorithm is given in Algorithm 3.1.

Algorithm 3.1 *Divide*
In: rectangle R, ordered set of dimensions corresponding to the long sides of R: $\{i_1, i_2, ..., i_m\}$
Out: $rectangleRefinement$, the set of sub-hyperrectangles

1: $rectangleRefinement \leftarrow \emptyset$
2: **for** $k = 1, 2, ..., m$ **do**
3: Divide the rectangle R along dimension i_k into thirds, and denote the resulting rectangles R_1, R_2, R_3. where R_2 is the center rectangle.
4: Assign each point in $knownPoints(R)$ to one of the containing rectangles (if in the intersection of 2 rectangles assign the point arbitrarily) in R_1, R_2, R_3
5: $rectangleRefinement \leftarrow rectangleRefinement \cup \{R_1, R_3\}$
6: $R \leftarrow R_2$
7: **end for**
8: $rectangleRefinement \leftarrow rectangleRefinement \cup \{R\}$

Note that when the given ordering satisfies $w_{i_1} \leq w_{i_2} \leq ... \leq w_{i_m}$, and $knownPoints(R) = \{center(R)\}$, then the algorithm is equivalent to the Divide algorithm of Jones et al. [12].

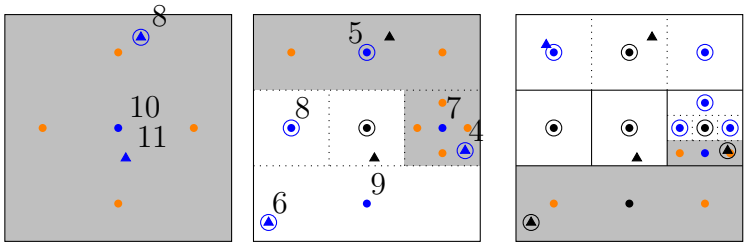
3.2 A simple extension to use external trial points (DUET-naive)

A minor modification to Algorithm 2.1 produces a new variant we call DUET-naive; the change is to update the rectangle function value with the minimum function value of all points contained in the rectangle. Algorithm 3.1, which assigns the lowest function value of all known points contained in the rectangle R, to $f(R)$. By setting $f(R)$ be the minimum function value over all $p \in knownPoints(R)$, we bias the search of DIRECT-naive to further explore hyperrectangles which contain low-valued points from external sources. Although DUET-naive does not change the general scheme of subdivision of hyperrectangles and sampling of *subpoints* of DIRECT, using the set of external trial points at each iteration, denoted as $additionalPoints$, affects the choice of potentially optimal hyperrectangles. DUET-naive is present in Algorithm 3.2.

Algorithm 3.2 DUET-naive
in: $evalPoints$, $additionalPoints$
out: $newPoints$

$ContainingRects \leftarrow \emptyset$
for all $p \in evalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$
  Save p as a *subpoint* of R.
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
  $ContainingRects \leftarrow ContainingRects \cup \{R\}$
end for
for all $p \in additionalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$.
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
end for
for all $R \in ContainingRects$ **do**
  Using *subpoints* order the longest sides of R, $i_1, i_2, ..., i_m$ such that $w_{i_1} \leq w_{i_2} \leq ... \leq w_{i_m}$
  $Rnew \leftarrow Divide(R, \{i_1, i_2, ..., i_m\})$
  $partition \leftarrow (partition \setminus \{R\}) \cup Rnew$
end for
$potOptimal \leftarrow FindPotOptimalRect(partition)$
for all $R \in potOptimal$ **do**
  $newPoints \leftarrow newPoints \cup \{p \in R | p$ is a subpoint of $R\}$
end for

Figure 3.1 shows the first three iterations of Algorithm 3.2. Notice that the choice of potentially optimal hyperrectangles differs from the corresponding iteration shown in Figure 2.1 because the smaller sized subrectangle to the right of the center now has a lower function value derived from an external trial point (with a function value of 4).



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 3.1. Example of the DUET-naive algorithm. The first three nontrivial iterations of the algorithm are shown. *evalPoints* and *newPoints* are shown as blue and orange points, respectively. *additionalPoints* are shown as blue triangles. Black points and triangles represent existing points (that have been *evalPoints* and *additionalPoints*, respectively, in a previous iteration). The current subdivision is shown as a dotted line and potentially optimal rectangles are shaded. The point with the lowest function value, in its containing rectangle R (and thus used to derive $f(R)$) is circled.

It becomes apparent in the example that we may be able to save on the evaluation of subpoints where there are already existing additional points. For example, notice that this is very close existing point to the subpoint that is slated for evaluation in the left side of the bottom rectangle in Iteration 3.

3.3 Random splitting to save on function evaluations (DUET-RS)

DIRECT uses subpoint values to order the (long) hyperrectangle sides to be subdivided as well as for setting $f(R)$ for each sub-hyperrectangle R that results from subdivision. We may be able to save on function evaluations at each iteration if we forego the ordering of dimensions according to the subpoint function values. Instead, in DUET-RS, we let the order of splits be random. This saves function evaluations when additional points already exist in the resulting sub-hyperrectangles, as shown in figure 3.2. Then we only need to evaluate center points for those new rectangles that do not contain any additional points.

The complete description in pseudo code of DUET-RS is given by Algorithm 3.3. DUET-RS subdivides the potentially optimal hyperrectangles and then only request to evaluate subpoints in hyperrectangles that have $knownPoints(R) = \emptyset$.

We show the first three nontrivial iterations of DUET-RS in Figure 3.3. In Iteration 1, we only need to evaluate three subpoints once we determine the subdivision into subrectangles, by making use of the top additional point. Since we some subpoints are note evaluated, not only the ordering of splits may change but also $f(R)$ for some of the rectangles. The bottom subrectangle in Iteration 2 and both potentially optimal rectangles in Iteration 3 only have one long dimension. In this case it is clear that there is only one possible ordering of splits.

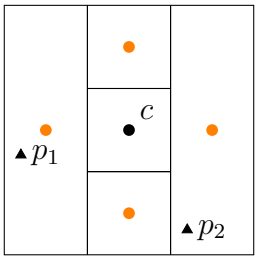
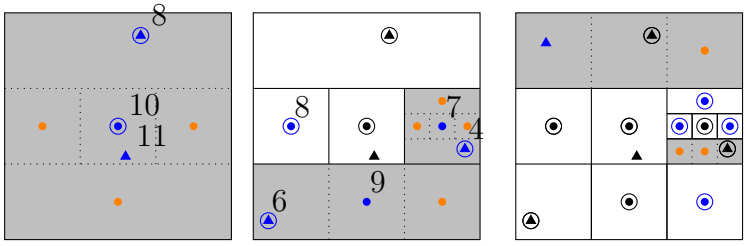


Figure 3.2. Example showing use of additional points by DUET-RS. Splitting of the rectangle is shown using solid lines. Subpoints are shown as orange dots. p_1 and p_2 are additional points that can each save on a function evaluation of the subpoint in their corresponding sub-hyperrectangles.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 3.3. Example of the DUET-RS algorithm. The first three nontrivial iterations of the algorithm are shown.

Algorithm 3.3 DUET-RS
in: $evalPoints$, $additionalPoints$
out: $newPoints$

for all $p \in evalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
end for
for all $p \in additionalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$.
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
end for
$potOptimal \leftarrow FindPotOptimalRect(partition)$
for all $R \in potOptimal$ **do**
  Generate a random ordering of the long sides of R, $\{i_1, i_2, ..., i_m\}$.
  $Rnew \leftarrow Divide(R, \{i_1, i_2, ...i_m\})$
  $partition \leftarrow (partition \setminus \{R\}) \cup Rnew$
  for all $R' \in Rnew$ **do**
    if $knownPoints(R') = \emptyset$ **then**
      $newPoints \leftarrow newPoints \cup \{center(R')\}$.
    end if
  end for
end for

3.4 Using subpoint estimates to save on function evaluations (DUET-SE)

The ordering of sides to be split in the DIRECT algorithm (i.e., in increasing order of w_i) has the desirable effect that dimensions that correspond to a lower minimum function value of their subpoints will be contained in larger subrectangles after subdivision. In the next iteration the larger rectangles that also have low function values will most likely be potentially optimal. This in turn will cause the rectangles with lower function values to be further subdivided. We would like to preserve the ordering of dimensions of DIRECT while making use of additional points to save on function evaluations when possible. We would like to estimate subpoints by trial points that are sufficiently close to subpoints. When an existing point is sufficiently far from a subpoint to be estimated, after subdivision, it is not clear that it will be contained in the same sub-hyperrectangle as the subpoint. A simple two dimensional example is shown in Figure 3.4.

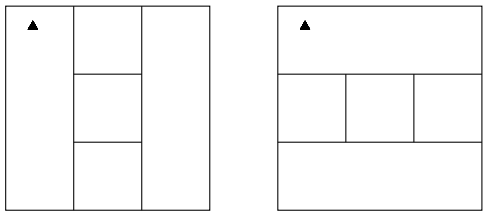


Figure 3.4. For the point in the top left corner, it is not clear in which rectangle the point will fall into, given that there are two possible distinct orderings for a hyperrectangle with two long sides.

Therefore, we wish to only consider additional points for which there is no ambiguity in knowing which subrectangle it will be in. As shown in Figure 3.5, the gray regions are unambiguous. The point p_1 will be in the same rectangle as the left subpoint no matter what order the rectangle is divided; we therefore say that $p_1 1$ is a *subpoint estimate*. We call the hypercube containing such points the *subpoint estimation region*.

The variant of DIRECT that uses subpoint estimates and additional points is given in Algorithm 3.4. The algorithm differs from Algorithm 2.1 in that it checks for each potentially optimal hyperrectangle and each of its subpoints whether it can be estimated by an existing point. In addition, since it is not necessarily the case that $center(R) \in knownPoints(R)$ for any given hyperrectangle R, if $knownPoints(C(R)) = \emptyset$, then $center(R)$ will also need to be evaluated and thus added to $newPoints$.

An example of the first three non-trivial iterations of the algorithm are shown in Figure 3.6. In Iteration 1, the top subpoint does not need to be evaluated since the top additional point lies within its corresponding subpoint estimation region. In Iteration 3, the subrectangle on the left does require any function evaluations because all of its subpoints have

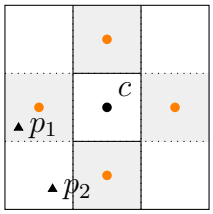
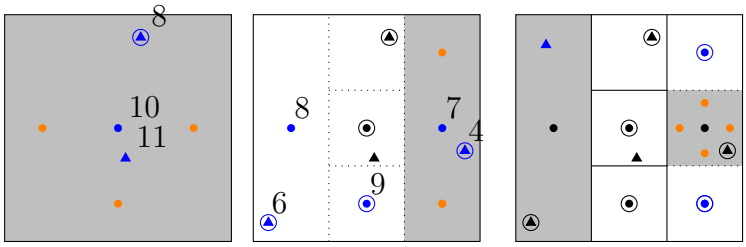


Figure 3.5. Subpoint estimation regions (hypercubes) are shown as the shaded rectangles. Subpoints are shown as orange dots, and additional points as triangles. Point p_1 is a subpoint estimate. Point p_2 cannot be a subpoint estimate (at this iteration) because it is not contained in a subpoint estimation region.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 3.6. Example of the DUET-SE algorithm. The first three nontrivial iterations of the algorithm are shown.

subpoint estimates. For the smaller subrectangle to the right of the center point, all four subpoints need to be evaluated because the existing additional point is not contained in any of the subpoint estimation regions.

Algorithm 3.4 DUET-SE
in: $evalPoints$, $additionalPoints$
out: $newPoints$

$ContainingRects \leftarrow \emptyset$
for all $p \in evalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$
  Save p as a *subpoint* of R
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
  $ContainingRects \leftarrow ContainingRects \cup \{R\}$
end for
for all $R \in ContainingRects$ **do**
  Order the longest sides of R, $i_1, i_2, ..., i_m$ such that $w_{i_1} \leq w_{i_2} \leq ... \leq w_{i_m}$
  $Rnew \leftarrow Divide(R, \{i_1, ..., i_m\})$
  $partition \leftarrow (partition \setminus \{R\}) \cup Rnew$
end for
for all $p \in additionalPoints$ **do**
  Find rectangle $R \in partition$ s.t. $p \in R$
  $knownPoints(R) \leftarrow knownPoints(R) \cup \{p\}$
end for
$potOptimal \leftarrow FindPotOptimalRect(partition)$
for all $R \in potOptimal$ **do**
  for all $p \in R$ s.t. p is a subpoint of R **do**
    if $p' \in knownPoints(R)$ and p' is a *subpoint estimate* of p **then**
      Save p' as a subpoint estimate of p
    else
      $newPoints \leftarrow newPoints \cup \{p\}$
    end if
    if $knownPoints(C(R)) = \emptyset$ **then**
      $newPoints \leftarrow center(R)$
    end if
  end for
end for

4 Empirical study

We report our empirical results for the Dixon dataset which contains nine different test functions. For each problem we run our algorithms with a number of $x \times n$ free external trial points, per iteration, where $x = 0, 1, 2, 3, 6, 10$. We report our statistics for the three different variants for each one of these settings over 20 runs. Note that if no points are added (i.e., $x = 0$), DUET-naive and DUET-SE are identical to DIRECT. We also report the Borda counts over the runs where we compare the three DUET variants and DIRECT. Each method is assigned a score in $\{4, 3, 2, 1\}$, corresponding to its rank (the first ranking being assigned the highest score). When two or more methods tie then they are assigned the median score of the ranks that are tied.

We show our experimental results in the following tables and figures. A summary is given in Table 4.10 where we summarize the empirical results in a table for all of the problem sets, the three variants of DUET and the DIRECT algorithm. The table shows the average and standard deviation of the number of evaluations normalized with respect to DIRECT. The Borda counts are also summarized over all of the test problems. Both measures show that DUET-SE outperforms DUET-naive and DUET-RS. DIRECT outperforms DUET-SE in terms of the average number of evaluations. However, the Borda count shows a clear advantage of DUET-SE. This implies that DUET-SE actually wins most of the time. When DUET-SE loses, it can perform quite poorly on some of the runs. This is also evident by the relatively high standard deviation of the (scaled) number of evaluations.

Table 4.1. Performance on the Branin function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	180						
	Borda Count	52	40	38	31	42	35	238
DUET-naive	Eval AVG	180	178.5	191.9	175.8	193.4	186.7	184.4
	Eval SD	0	21.46	61.12	41.7	67.96	56.69	47.37
	Free AVG	0	32.6	70	99.3	215.4	350	127.9
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.9917	1.066	0.9767	1.074	1.037	1.024
	Scaled Eval SD	0	0.1192	0.3395	0.2316	0.3775	0.3149	0.2631
	Borda Count	52	45.5	41	41	41	39.5	260
DUET-RS	Eval AVG	189.2	180.2	160.9	142	154.9	118.2	157.6
	Eval SD	25.88	31.46	54.39	50.14	67.01	56.94	54.14
	Free AVG	0	31.5	61.6	93.9	218.4	328	122.2
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1.051	1.001	0.8942	0.7889	0.8608	0.6569	0.8755
	Scaled Eval SD	0.1438	0.1748	0.3022	0.2785	0.3723	0.3163	0.3008
	Borda Count	44	47.5	56	64.5	59	63.5	334.5
DUET-SE	Eval AVG	180	166.4	170.6	145.2	152.4	136.1	158.4
	Eval SD	0	21.28	65.71	39.4	50.76	59.28	47.07
	Free AVG	0	32.8	70	100.2	223.8	374	133.5
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.9247	0.9475	0.8064	0.8467	0.7561	0.8802
	Scaled Eval SD	0	0.1182	0.3651	0.2189	0.282	0.3293	0.2615
	Borda Count	52	67	65	63.5	58	62	367.5

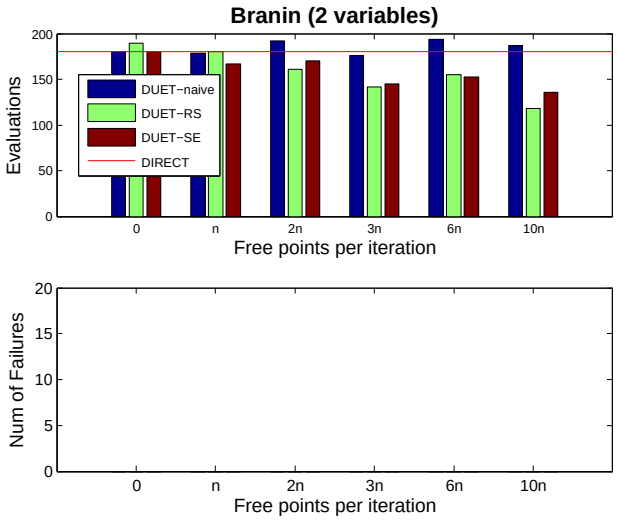


Figure 4.1. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Branin dataset.

Table 4.2. Performance on the Shekel 5 function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	152						
	Borda Count	60	41.5	55	46.5	54.5	59	316.5
DUET-naive	Eval AVG	152	270	656.7	641	495.7	442.4	421.8
	Eval SD	0	524.5	1332	1355	829.4	617	878.6
	Free AVG	0	119.4	899.7	624	611.1	1214	578.1
	Runs > 5000	0	1	5	4	7	6	23
	Scaled Eval AVG	1	1.776	4.32	4.217	3.261	2.911	2.775
	Scaled Eval SD	0	3.451	8.765	8.915	5.457	4.059	5.781
	Borda Count	60	53.5	46.5	42.5	40.5	45	288
DUET-RS	Eval AVG	1535	844.8	1431	552.9	479.9	1148	1041
	Eval SD	841.3	891.5	1430	660.1	502	1582	1075
	Free AVG	0	635.7	3177	1744	5760	2571	2315
	Runs > 5000	1	5	9	7	10	9	41
	Scaled Eval AVG	10.1	5.558	9.416	3.638	3.157	7.553	6.846
	Scaled Eval SD	5.535	5.865	9.411	4.343	3.303	10.41	7.072
	Borda Count	20	31	33	40.5	37.5	37	199
DUET-SE	Eval AVG	152	251	484.9	279.6	382.7	414.8	314.2
	Eval SD	0	493.3	937.7	455.4	600.9	650	566.8
	Free AVG	0	121.1	902.4	482.1	971.3	1298	629.2
	Runs > 5000	0	1	5	3	3	7	19
	Scaled Eval AVG	1	1.651	3.19	1.839	2.518	2.729	2.067
	Scaled Eval SD	0	3.245	6.169	2.996	3.953	4.276	3.729
	Borda Count	60	74	65.5	70.5	67.5	59	396.5

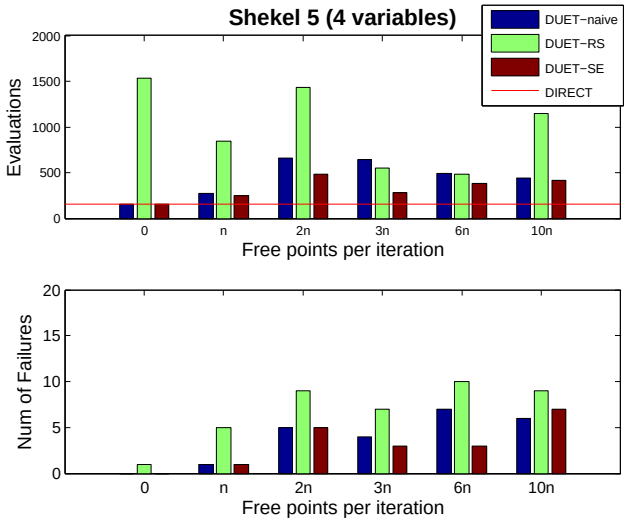


Figure 4.2. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Shekel 5 dataset.

Table 4.3. Performance on the Shekel 7 function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	142						
	Borda Count	60	49	51.5	56	59	56	331.5
DUET-naive	Eval AVG	142	141.8	141.7	185.7	369.1	503.6	237.9
	Eval SD	0	0.8944	1.237	119.1	574	836.4	409.8
	Free AVG	0	64	128	231	952.5	1139	419.1
	Runs > 5000	0	0	2	0	4	3	9
	Scaled Eval AVG	1	0.9986	0.9977	1.308	2.599	3.547	1.675
	Scaled Eval SD	0	0.006299	0.008709	0.8387	4.042	5.89	2.886
	Borda Count	60	51	50	48	45.5	40.5	295
DUET-RS	Eval AVG	1340	1138	1284	748.5	1359	1149	1193
	Eval SD	1171	1324	1320	669	1357	1644	1286
	Free AVG	0	578.2	1003	1342	4576	3875	1895
	Runs > 5000	8	11	9	10	3	5	46
	Scaled Eval AVG	9.439	8.011	9.045	5.271	9.573	8.092	8.401
	Scaled Eval SD	8.249	9.327	9.295	4.711	9.56	11.57	9.057
	Borda Count	20	20	21.5	20	31	38	150.5
DUET-SE	Eval AVG	142	127.4	140.3	226.3	283.6	198.1	182.9
	Eval SD	0	3.218	88.85	421.3	460.8	303.3	276.8
	Free AVG	0	64	138.5	273.6	1028	872.9	396.1
	Runs > 5000	0	0	1	0	4	3	8
	Scaled Eval AVG	1	0.8972	0.9881	1.594	1.997	1.395	1.288
	Scaled Eval SD	0	0.02266	0.6257	2.967	3.245	2.136	1.949
	Borda Count	60	80	77	76	64.5	65.5	423

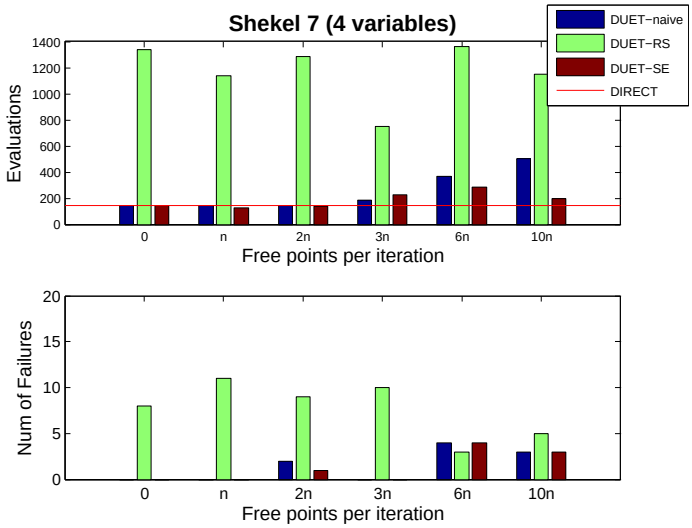


Figure 4.3. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Shekel 7 dataset.

Table 4.4. Performance on the Shekel 10 function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	142						
	Borda Count	60	53	55.5	61.5	62	66.5	358.5
DUET-naive	Eval AVG	142	142	142.2	501.3	261.9	1133	365.9
	Eval SD	0	0.686	0.6831	791.4	355.5	1398	716.1
	Free AVG	0	64	128	550.4	624	2677	674
	Runs > 5000	0	2	4	5	6	5	22
	Scaled Eval AVG	1	1	1.002	3.531	1.844	7.982	2.577
	Scaled Eval SD	0	0.004831	0.004811	5.573	2.503	9.845	5.043
	Borda Count	60	48	45	39	38	33.5	263.5
DUET-RS	Eval AVG	857.6	1298	1116	1017	868.3	603.4	947.3
	Eval SD	652.9	733.6	1375	692.3	505.7	383.4	779.9
	Free AVG	0	696.6	1124	3234	5874	5468	2733
	Runs > 5000	5	7	7	4	7	4	34
	Scaled Eval AVG	6.039	9.143	7.858	7.164	6.115	4.249	6.671
	Scaled Eval SD	4.598	5.166	9.681	4.875	3.562	2.7	5.493
	Borda Count	20	24	25	39	31	43	182
DUET-SE	Eval AVG	142	127.6	142.6	601.9	356.3	801.8	343.2
	Eval SD	0	2.615	76.85	983.4	599.8	1266	697.8
	Free AVG	0	64	144	1549	944	3362	1011
	Runs > 5000	0	2	1	4	2	4	13
	Scaled Eval AVG	1	0.8987	1.004	4.239	2.509	5.647	2.417
	Scaled Eval SD	0	0.01842	0.5412	6.925	4.224	8.913	4.914
	Borda Count	60	75	74.5	60.5	69	57	396

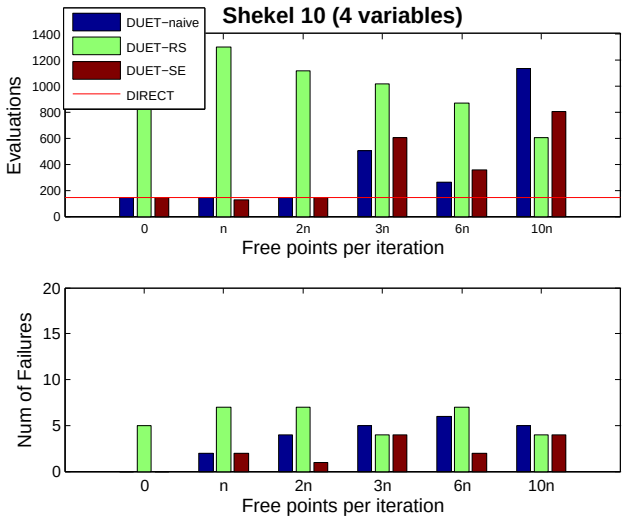


Figure 4.4. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Shekel 10 dataset.

Table 4.5. Performance on the Hartman 3 function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	179						
	Borda Count	59	43	56.5	51	57	57	323.5
DUET-naive	Eval AVG	179	176.2	198.9	210.6	267.4	290.2	220.4
	Eval SD	0	2.546	98.43	110	175.2	195.5	128
	Free AVG	0	45	95.1	148.9	333.9	582	200.8
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.9844	1.111	1.177	1.494	1.621	1.231
	Scaled Eval SD	0	0.01423	0.5499	0.6145	0.9786	1.092	0.715
	Borda Count	59	55	59.5	56	45.5	41	316
DUET-RS	Eval AVG	498.9	366.2	363.2	435.5	327.8	290.4	380.3
	Eval SD	185.6	152.2	120.9	169.8	163.6	126.4	166.6
	Free AVG	0	64.5	136.2	231.3	413.1	667.5	252.1
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	2.787	2.046	2.029	2.433	1.831	1.622	2.125
	Scaled Eval SD	1.037	0.8505	0.6754	0.9486	0.9137	0.7064	0.9305
	Borda Count	23	29	35	30	41.5	44.5	203
DUET-SE	Eval AVG	179	191.8	327.7	240.2	279.9	247.8	244.4
	Eval SD	0	86.81	167.3	141.1	161.9	136.2	136.2
	Free AVG	0	48.9	138.3	174.6	404.1	658.5	237.4
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	1.071	1.831	1.342	1.564	1.384	1.365
	Scaled Eval SD	0	0.485	0.9344	0.7885	0.9044	0.7609	0.7612
	Borda Count	59	73	49	63	56	57.5	357.5

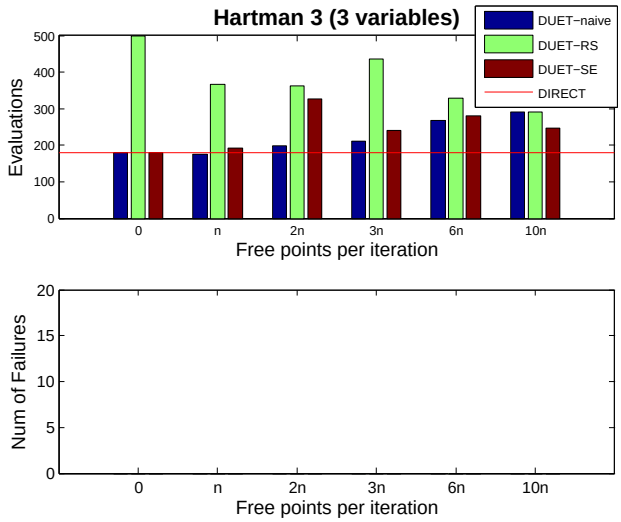


Figure 4.5. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Hartman 3 dataset.

Table 4.6. Performance on the Hartman 6 function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	528						
	Borda Count	60	70	75	75	78	77	435
DUET-naive	Eval AVG	528	2804	3217	2596	3038	2172	2376
	Eval SD	0	1530	1340	1615	1349	1139	1548
	Iter AVG	NaN	87.15	98.7	81.21	89.12	67.37	0
	Free AVG	0	522.9	1184	1462	3208	4042	1737
	Runs $>$ 5000	0	0	0	1	3	1	5
	Scaled Eval AVG	1	5.31	6.094	4.917	5.753	4.113	4.5
	Scaled Eval SD	0	2.897	2.538	3.059	2.555	2.156	2.932
	Borda Count	60	45	39	33.5	36	36	249.5
DUET-RS	Eval AVG	2969	3547	2780	2195	3190	2312	2846
	Eval SD	853.5	841.3	1401	1360	1081	1243	1198
	Iter AVG	NaN	93.11	77.27	68.36	98.23	79.62	0
	Free AVG	0	558.7	927.3	1231	3536	4778	1838
	Runs $>$ 5000	9	11	9	9	7	12	57
	Scaled Eval AVG	5.624	6.718	5.265	4.158	6.043	4.379	5.39
	Scaled Eval SD	1.616	1.593	2.653	2.576	2.047	2.355	2.27
	Borda Count	20	24.5	33	33.5	34	30	175
DUET-SE	Eval AVG	528	2289	3024	2464	2873	1898	2169
	Eval SD	0	1568	1323	1530	1273	1108	1482
	Iter AVG	NaN	75.89	96.6	83.35	92.28	67.79	0
	Free AVG	0	455.4	1159	1500	3322	4067	1751
	Runs $>$ 5000	0	1	0	0	2	1	4
	Scaled Eval AVG	1	4.335	5.728	4.666	5.442	3.594	4.108
	Scaled Eval SD	0	2.969	2.505	2.898	2.412	2.098	2.806
	Borda Count	60	60.5	53	58	52	57	340.5

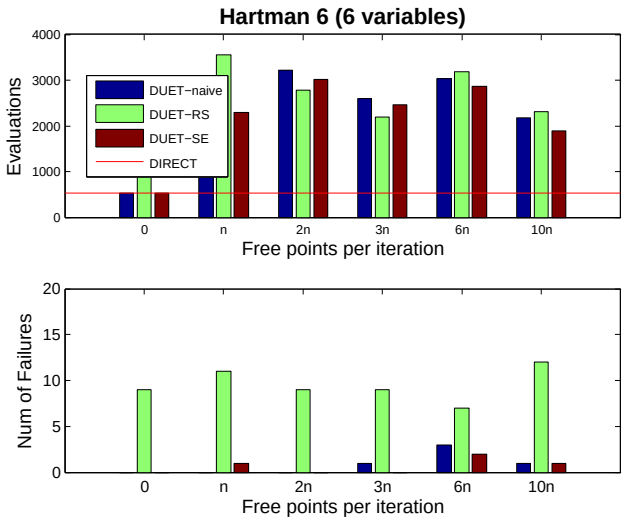


Figure 4.6. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Hartman 6 dataset.

Table 4.7. Performance on the GP function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	166						
	Borda Count	41	27	22.5	21	20	20	151.5
DUET-naive	Eval AVG	166	158.1	151.5	151.1	144	140.1	151.8
	Eval SD	0	9.894	8.205	10.29	10.58	14.03	12.9
	Free AVG	0	29.7	57.6	87.6	168	274	102.8
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.9524	0.9127	0.9102	0.8675	0.844	0.9145
	Scaled Eval SD	0	0.0596	0.04943	0.06199	0.06375	0.08452	0.07773
	Borda Count	41	36	38.5	41	41	41	238.5
DUET-RS	Eval AVG	123.9	109.5	100.7	92.15	71.35	59.6	92.87
	Eval SD	22.32	22.97	20.71	19.43	23.15	13.69	29.84
	Free AVG	0	24.7	47.2	72.9	137.4	231	85.53
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	0.7464	0.6599	0.6063	0.5551	0.4298	0.359	0.5594
	Scaled Eval SD	0.1345	0.1384	0.1247	0.117	0.1394	0.08249	0.1798
	Borda Count	77	79	79	77	77	80	469
DUET-SE	Eval AVG	166	149.2	133.3	128	104	92.65	128.9
	Eval SD	0	13.74	10.49	14.8	11.95	16.8	27.93
	Free AVG	0	30.3	58	87.9	174	288	106.4
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.8991	0.803	0.7711	0.6262	0.5581	0.7763
	Scaled Eval SD	0	0.08276	0.06318	0.08916	0.07198	0.1012	0.1683
	Borda Count	41	58	60	61	62	59	341

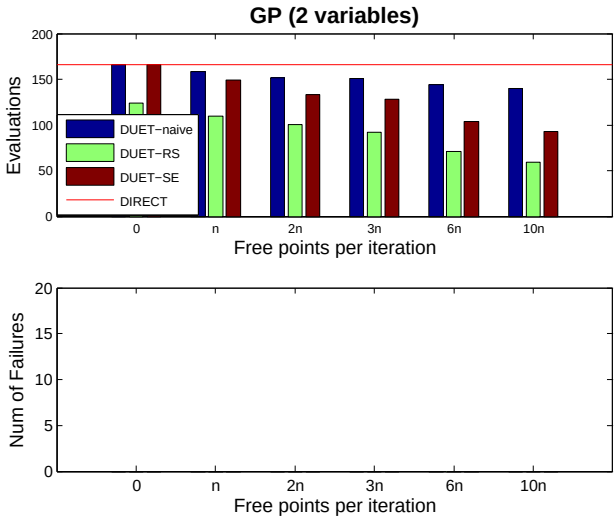


Figure 4.7. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the GP dataset.

Table 4.8. Performance on the Camel function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	246						
	Borda Count	44	28	21	20	20	20	153
DUET-naive	Eval AVG	246	181	162.9	148.8	141.8	132.5	168.8
	Eval SD	0	64.95	45.33	36.88	40.81	27.11	55.18
	Free AVG	0	28.6	59.4	89.7	173.4	300	108.5
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.7358	0.6622	0.6049	0.5766	0.5386	0.6863
	Scaled Eval SD	0	0.264	0.1843	0.1499	0.1659	0.1102	0.2243
	Borda Count	44	45.5	49	50	44.5	40	273
DUET-RS	Eval AVG	227.2	156.1	142.3	129.9	93.5	74.65	137.3
	Eval SD	21.4	36.56	47.4	44.43	31.56	24.23	60.21
	Free AVG	0	28.9	59.6	91.8	178.2	281	106.6
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	0.9234	0.6346	0.5787	0.5283	0.3801	0.3035	0.5581
	Scaled Eval SD	0.087	0.1486	0.1927	0.1806	0.1283	0.09851	0.2447
	Borda Count	68	58.5	64	62	67.5	69	389
DUET-SE	Eval AVG	246	160.8	138.2	122.3	98.1	75.95	140.2
	Eval SD	0	62.83	36.08	33.37	32.63	25.81	65.5
	Free AVG	0	29.2	61.6	92.7	183	309	112.6
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.6535	0.562	0.4972	0.3988	0.3087	0.57
	Scaled Eval SD	0	0.2554	0.1467	0.1356	0.1327	0.1049	0.2663
	Borda Count	44	68	66	68	68	71	385

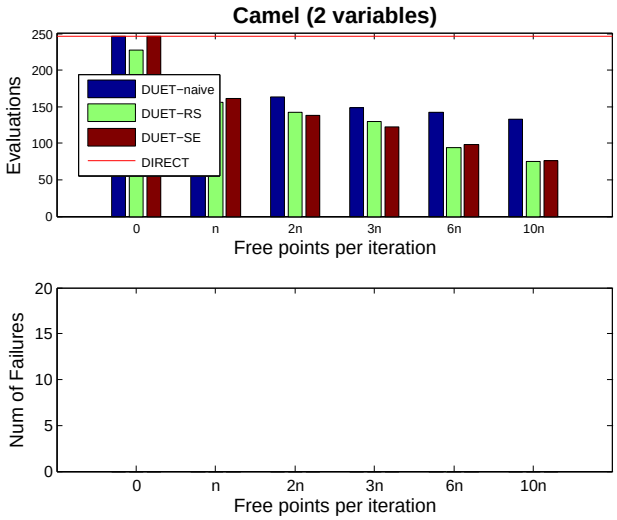


Figure 4.8. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Camel dataset.

Table 4.9. Performance on the Shubert function.

	Random Pt Num.	0n	1n	2n	3n	6n	10n	Total
DIRECT	Evals	2957						
	Borda Count	50	21	20	20	20	20	151
DUET-naive	Eval AVG	2957	769.1	543.2	332.8	208.7	172.8	830.6
	Eval SD	0	764.8	420.6	188.5	102.2	92.13	1041
	Free AVG	0	154	242.8	273	411.6	606	281.2
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1	0.2601	0.1837	0.1125	0.07058	0.05844	0.2809
	Scaled Eval SD	0	0.2586	0.1422	0.06375	0.03456	0.03116	0.3521
	Borda Count	50	56	48	52	47	44	297
DUET-RS	Eval AVG	2749	871.4	345.7	355.6	155.7	104.5	763.6
	Eval SD	144.5	772.5	267.9	279.5	108.1	63.95	990.8
	Free AVG	0	238.5	226	486.9	651	758	393.4
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	0.9295	0.2947	0.1169	0.1203	0.05264	0.03532	0.2582
	Scaled Eval SD	0.04886	0.2612	0.09062	0.09451	0.03655	0.02163	0.3351
	Borda Count	77	58.5	69	61	66	68	399.5
DUET-SE	Eval AVG	2959	781.6	374.3	296.4	153.6	98.15	777.1
	Eval SD	0.6156	724.7	239.1	215	93.99	61.77	1054
	Free AVG	0	191.2	266.8	363.3	586.8	723	355.2
	Runs > 5000	0	0	0	0	0	0	0
	Scaled Eval AVG	1.001	0.2643	0.1266	0.1002	0.05193	0.03319	0.2628
	Scaled Eval SD	0.0002082	0.2451	0.08087	0.0727	0.03178	0.02089	0.3565
	Borda Count	23	64.5	63	67	67	68	352.5

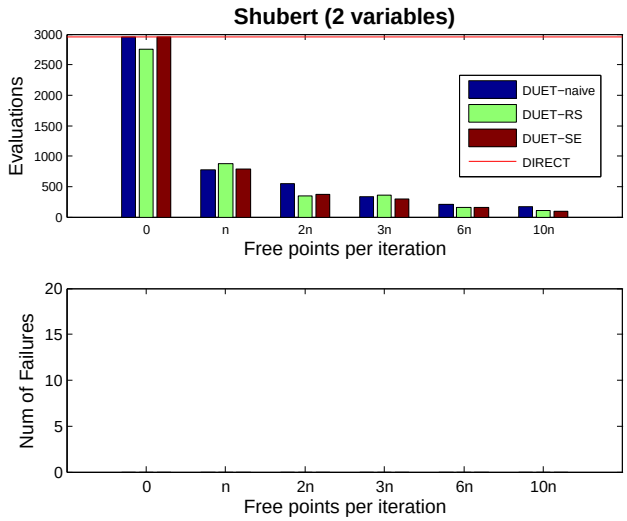


Figure 4.9. A plot showing the average number of evaluations and total number of failures with respect to the number of (randomly generated) points added at each iteration for the Shubert dataset.

Table 4.10. A summary of the empirical results, comparing the variants of DUET and the DIRECT algorithm. The average number of function evaluations is normalized so that the number of evaluations required by DIRECT is one.

	Relative Avg Evals (Std Dev)	**Num of Failures**	**Borda Count**
DIRECT	1.00 (0)	0	2458.5
DUET-naive	1.69 (3.01)	59	2480.5
DUET-RS	2.88 (4.79)	178	2501.5
DUET-SE	1.49 (2.54)	44	3359.5

This page intentionally left blank.

5 Conclusion

We have compared our three variants of DUET which extend the DIRECT algorithm to use external trial points. DUET-SE maintains the ordering of splits while making use of free function evaluations of external trial points. We have found that, overall, DUET-SE outperforms the two other variants: DUET-naive and DUET-RS. DUET-SE outperforms DUET-naive because it directly saves on function evaluations of subpoints when external points are available can be used as subpoint estimates. DUET-RS does not perform very well, indicating that the order of splits significantly contributes to the efficiency of the DIRECT algorithm.

When optimizing in parallel and using external points that are generated by other optimization algorithms, we hope that the performance of the DUET variants should improve as compared with using randomly generated points. When using external trial points that are generated by local optimization algorithms then this may introduce a local bias in DIRECT which may require fine tuning of the algorithm (e.g., adjustment of ϵ parameter). This remains a topic for future investigation.

There are several related topics for future research. The possibility of sampling hyperrectangles at points different than the center point opens up the possibility of sampling at points that may help to speed converge to solutions that lie on the boundary. In practice, solutions often lie on a constraint boundary. We can potentially enlarge to initial partition to be larger then the feasible region; then, when the center point is infeasible we evaluate the closest feasible point to the center. Another topic for future research would be to investigate whether the theoretical justification for Lipschitz continuous functions can be preserved by sampling only at center points but using additional points to strengthen the bounds for Lipschitz continuous functions. Since the convex hull computation (implicitly) implies a lower and upper bound on the Lipschitz constant for each hyperrectangle on the lower convex hull, these bounds may be used to further compute bounds on the objective function.

References

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

[2] R. G. Carter, J. M. Gablonsky, A. Patrick, C. T. Kelley, and O. J. Eslinger. Algorithms for noisy problems in gas transmission pipeline optimization. *Optim. Eng*, 2:139–157, 2001.

[3] Steven E. Cox, William E. Hart, Raphael Haftka, and Layne Watson. DIRECT algorithm with box penetration for improved local convergence. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002. AIAA-2002-5581.

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

[5] D. E. Finkel and C. T. Kelley. Convergence analysis of the DIRECT algorithm. Technical Report CRSC-TR04-28, CSRC, NSCU, 2004.

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

[7] J. M. Gablonsky. Direct version 2.0 user guide technical report crsc-tr01-08. Technical report, Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, 2001.

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

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

[10] P. Hansen and B. Jaumard. Lipschitz optimization. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, pages 407–493. Kluwer Academic Publishers, 1995.

[11] Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, June 1999.

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

[13] Donald R. Jones. Direct global optimization algorithm. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 431–440. Kluwer Academic Publishers, 2001.

[14] Mark T. Jones and Merrell L. Patrick. The use of Lanczos's method to solve the large generalized symmetric definite eigenvalue problem. Technical Report NASA CR-181914, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA, September 1989.

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

[16] Chiter Lakhdar. Towards a new direct algorithm: a two-points based sampling method. `http://www.optimization-online.org/DB_FILE/2005/03/1077.pdf`, 2005.

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