Optimization Methods & Software
Vol. 27, No. 6, December 2012, 1059–1072

Taylor & Francis
Taylor & Francis Group

A cover partitioning method for bound constrained global optimization

C.J. Price*, M. Reale and B.L. Robertson

Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand

(*Received 9 May 2010; final version received 18 January 2011*)

A stochastic algorithm for global optimization subject to simple bounds is described. The method is applicable to black-box functions which may be non-smooth or discontinuous. The algorithm is in the spirit of the deterministic algorithm DIRECT of Jones, Perttunen, and Stuckman. Like DIRECT, it generates successively finer covers of the feasible region, where each cover consists of a finite number of boxes, and each box is defined by simple bounds. Its principal difference is that it calculates the objective at a randomly selected point in each unpopulated box, rather than at the centre of the box. A limited storage version of the algorithm is also presented. The sequence of best-known function values is shown to converge to the essential minimum with probability 1 for both versions of the algorithm. A worst case expected rate theorem is established. Numerical results are presented which show the methods are effective in practice.

Keywords: direct search; dividing rectangles; stochastic global optimization; tilecutter

1. Introduction

The bound constrained global optimization problem is of the form

$$\min_{x\in\Omega} f(x), \tag{1}$$

where Ω is the box defined by

$$\Omega = \{x \in R^n : \ell_i \le x_i \le u_i \quad \forall i = 1, \ldots, n\}. \tag{2}$$

The objective function f maps Ω into $R \cup \{+\infty\}$. The objective function is assumed to be a Lebesgue measurable black-box function without further defining properties. For convenience, f is given the value $+\infty$ at places where it cannot be evaluated. This allows functions which are defined on some bounded region to be extended to a box of the form Ω. In particular, constraints which are not simple bounds can be handled by the extreme barrier approach [6] which defines $f = +\infty$ whenever any such constraint is violated. Alternatively, global optimization problems with general constraints could be solved using an exact penalty function or an augmented

*Corresponding author. Email: c.price@math.canterbury.ac.nz

ISSN 1055-6788 print/ISSN 1029-4937 online

http://dx.doi.org/10.1080/10556788.2011.557726
http://www.tandfonline.com

Lagrangian approach [3,4,7] provided the objective and constraints satisfy any relevant continuity or smoothness conditions.

The lack of any special properties for f such as smoothness, continuity, or a Lipschitz constant severely restricts how (1) can be attacked if convergence is to be obtained in any sense. Nevertheless a variety of stochastic and deterministic methods exist such as pure random search (PRS) [17], adaptive random search, and DIRECT [10]. PRS simply evaluates the function at a number of randomly chosen sample points and returns the lowest point as the estimate of the global minimizer. Developments of PRS seek to direct attention to parts of Ω considered more promising, based on information from previous function values. For example, [13] regularly modifies the probability distribution from which the sample points are drawn. In contrast, adaptive random search [12,16] attempts to exploit information from early sample points by partitioning Ω into subregions, and randomly sampling from each subregion. Those subregions which the early sample points suggest are lower are favoured with more sample points than others.

Recursive random search (RRS) [18] uses a strategy similar in spirit to the clustering algorithms of Rinnooy Kan and Timmer [14,15]. The latter generates sample points randomly in Ω and groups the lowest 20% of sample points into clusters. One local search is done from the lowest point in each cluster. RRS generates a number of random sample points, and the lowest of these defines a level. A local search is then performed from sample points which are generated randomly in Ω and lie below this level. The local search is conducted by sampling in a neighbourhood of a low sample point. Each time that sample point is improved, the sample point is replaced by the improving point. If no improvement is made after a finite number of samples, the neighbourhood is shrunk. This process continues until the neighbourhood reaches a minimum size, after which the local search is abandoned, and a new low sample point is sought. This strategy of alternating between local and global searches means that RRS makes good improvements early on.

The strategy behind accelerated random search (ARS) [5] is somewhat different. It randomly samples in a finite contracting sequence of subregions centred on the best-known point. Once a new best point is found, or the sequence is exhausted it goes back to the start of the sequence. The first region in this sequence is necessarily Ω.

The DIRECT algorithm of Jones *et al.* [10] is a deterministic method of solving (1) and (2). DIRECT generates a sequence of successively finer covers of Ω. Each cover consists of a finite number of boxes (or tiles) and f is calculated at the centre point of each box. The first cover consists of one box: Ω. At each iteration one or more boxes are cut into three equal parts by two hyperplanes perpendicular to one of the coordinate axes. Cutting into three parts means the centre point of the old box is also the centre point of one of the new boxes, saving a function evaluation.

DIRECT has certain disadvantages. It requires the storage of every function value and box, so storage and processing overheads grow faster than the number of function evaluations. This cannot be dodged by restarting since DIRECT, being deterministic, regenerates all the known function values before generating new ones. Secondly, when a box is cut, the centres of the three new boxes differ in only one coordinate. If, for example, the function is nearly constant along that coordinate direction, then two of those function evaluations are wasted. To avoid these undesirable features, we seek a stochastic version of the DIRECT algorithm.

Stochastic methods can have advantages over deterministic ones in semi-infinite programming [8]. These are problems of the form

$$\min_{x \in R^n} f_{\text{sip}}(x) \quad \text{such that } g_{\text{sip}}(x, t) \geq 0 \quad \forall t \in \Omega$$

where Ω can be of the form (2). Semi-infinite problems can be solved iteratively, where each iteration requires finding the global minimizers of $g_{\text{sip}}(x, t)$ over $t \in \Omega$, for a fixed value of x. If the method used to solve this subproblem is restricted to a predetermined finite set of points (as is the case with DIRECT in practice if the number of function evaluations per iteration is bounded)

then the iterative process used to solve the semi-infinite problem can fail. What happens is that the global minimizers may be forced into the gaps between the finite set of points the global optimization method is restricted to [8]. Stochastic methods have no connection between the sets of points sampled in different executions, and so do not suffer from this deficiency.

2. The algorithm

The algorithm is in the spirit of DIRECT [10]. It generates a sequence of successively finer covers of the feasible region Ω. Each cover consists of a finite number of tiles, where a tile is a region of the form

$$\{x \in R^n : a_i \leq x_i \leq b_i \quad \forall i = 1, \ldots, n\},$$

where the values of a_i and b_i vary from tile to tile. One sample point is placed randomly within each tile. The method iteratively selects a number of tiles, and cuts each selected tile into two smaller tiles. A tile is selected for cutting via its size and the function value at the sample point it contains. This iterative process generates a sequence of best-known points $\{x^{(k)}\}$ with an associated monotonically decreasing sequence of function values $\{f^{(k)}\}$. Here k is the iteration counter. The basic structure of TILECUTTER is as follows.

ALGORITHM 1

(1) *Randomly generate one initial point $x_1 \in \Omega$. Set $k = 1$.*
(2) *Select tiles to be cut, including at least one largest tile.*
(3) *Cut each selected tile in two, and randomly select a sample point in each new tile without a sample point. Delete all tiles which have just been cut into two new tiles.*
(4) *Let $x^{(k)}$ be the best-known point, and set $f^{(k)} = f(x^{(k)})$.*
(5) *If the maximum number of tiles is reached, delete all tiles and sample points. Choose a small number of new tiles, one of which must be Ω, and calculate f at one random sample point in each new tile.*
(6) *Check stopping conditions. If not satisfied, increment k and go to step* 2.

The main steps of the algorithm are described in detail in the following three subsections.

2.1 *Selection strategies*

There is much freedom in which tiles are selected for subdivision at each iteration. Tiles may be selected using a number of features, of which two important ones are the size and (nominal) height of the tile. The size of a tile T is the 1-norm of the diagonal of the tile, and this size is given the notation $\|T\|_1$. The nominal height is the function value in the tile calculated since the last restart. The actual height of the tile varies from point to point. The actual height is unknown, so the nominal height is used as a representative value and is simply called the height of the tile from now on. For convergence purposes, the selection strategy must satisfy the following assumption, no other conditions are needed.

ASSUMPTION 2.1 *At each iteration at least one tile for which $\|T\|_1$ is maximal is subdivided.*

Our selection strategy is based on the following principles. For tiles of a given size, the lower that tile, the more likely it is to contain a global minimizer. For tiles of the same height, the larger the tile, the more likely it is to contain a global minimizer. If one tile is both larger and higher than

another, it is not obvious which tile is more likely to contain a global minimizer. In generating the numerical results, all Pareto optimal tiles which exceed a minimum size τ_{acc} are selected. Here a tile T is Pareto optimal if every other tile is either smaller or higher than T. If two tiles have the same size and height, the tile generated first is considered to be the Pareto optimal tile of the pair. If limited storage space permits the cutting of only some selected tiles within an iteration, then the tiles are cut in decreasing order of size.

2.2 *Cutting a tile*

A tile T defined by the bounds $a \leq x \leq b$ is cut as follows. The tile is cut orthogonally to its longest edge. If more than one longest edge exists, the one with the lowest index is chosen. This cut divides the tile into two smaller tiles. It is not compulsory that the two new tiles be the same size. However, the larger can be at most A times the Lebesgue measure of the smaller, where $A \geq 1$. This restriction means when a tile is subdivided, both resulting sub-tiles are significantly smaller than the original tile. When $A > 1$, the cut is placed randomly within the permitted interval in which the larger tile is at most A times the Lebesgue measure of the smaller. Hence applying the algorithm twice yields different tilings each time, even if the tiles are subdivided in exactly the same order both times. The sample point from the tile T lies within one of the two new tiles. A sample point is randomly selected in the other new tile, and f is evaluated at this point.

The algorithm has significantly different characteristics depending on whether $A = 1$ or $A > 1$. If $A = 1$ any two tiles that have been cut from Ω using k cuts are *exactly* the same size. Hence if no tile has been generated using more than k cuts, at most k tiles can be Pareto optimal. In contrast, if $A > 1$ then every tile will be of a different size, with probability 1. This allows many more than k tiles to be Pareto optimal. The $A > 1$ version cuts up the larger tiles more quickly, and so focuses more on searching globally than the $A = 1$ version. The latter puts more effort into local search(es) around the best-known point(s), at the expense of exploring large, high tiles. In this respect, the $A = 1$ algorithm is much more like DIRECT than its $A > 1$ counterpart.

In the case when Ω is, or has been scaled to the unit hypercube, the choice of which edge to cut becomes equivalent to the least reduced axis rule [1].

2.3 *Restarting*

One feature of both TILECUTTER and DIRECT is that they store every sample point along with its function value. With DIRECT, these increasing storage and overhead requirements cannot be avoided by restarting because DIRECT is a deterministic algorithm. Restarting it regenerates all known function values before generating any new information, and is thus pointless. In contrast, the stochastic nature of our algorithm means restarting generates different tilings and sets of sample points each time, and is thus a valid strategy for reducing storage and overheads. At each restart one must start with a tiling which covers Ω.

It is possible to incorporate information from previous executions of TILECUTTER. A simple method of doing so is to choose a small number of tiles, one of which must be Ω. These tiles can overlap one another, if desired. Hence one could choose a limited number of tiles of different sizes all centred on the best-known point. Cutting Ω would continue the global search, whereas sample points generated in cutting other smaller initial tiles would execute more localized searches. Of course, if the best-known point were nowhere near the global minimizer, then this strategy would be counter-productive. The authors conducted limited numerical tests with such a strategy. Results showed that the algorithm was able to refine the lowest known minimizer more quickly, but it was also prone to mistaking a local minimizer for a global minimizer. Since our interest here is in global optimization, this strategy was not pursued further. Herein each restart begins with one tile: Ω.

3. Convergence results

In this section, we show that TILECUTTER is convergent with or without restarts. The probability of generating a point inside a chosen level set of f is established for the worst case choice of f. It is shown that this worst case result is independent of restarts provided that each restart uses only Ω as the sole initial tile. First, we state a precise definition of an acceptable solution to (1) and (2).

DEFINITION 3.1 *The essential minimum f^* on Ω is the supremum of the values $f_0 \in R$ for which the level set*

$$L(f_0) = \{x \in \Omega : f(x) \leq f_0\}$$

has a Lebesgue measure zero.

This definition is theoretically important, especially for exact arithmetic implementations of the algorithm. The following variant of the concept is also useful, particularly for finite precision implementations of the method.

DEFINITION 3.2 *The open set essential minimum $f^\sharp$ on Ω is the supremum of the values $f_0 \in R$ for which the level set*

$$L(f_0) = \{x \in \Omega : f(x) \leq f_0\}$$

contains no open ball of positive radius.

Clearly $f^* \leq f^\sharp$. An algorithm which runs forever can only calculate f at a countable set S of points, and so there is no guarantee that the essential minimum f^* will be attained in the limit. Provided that S is dense in Ω, the open set essential minimum $f^\sharp$ will be attained in the limit. The definition of $f^\sharp$ means that there is a dense set in Ω on which f is not less than $f^\sharp$. Hence $f^\sharp$ is the best value one can *guarantee* for a black box function if the algorithm is run forever.

We now show the sequence of iterates, without restarts, is dense in Ω. In the following analysis, we assume the algorithm is run forever in exact arithmetic.

PROPOSITION 3.3 *Let T_m be a tile generated by subdividing Ω m times. If, after every cut, the ratio of measures of the two new tiles is at most A, then*

(1)

$$\|T_m\|_1 \leq \left(1 - \frac{1}{(1+A)n}\right)^m \|\Omega\|_1,$$

where $\|T_m\|_1$ is the 1-norm of the diagonal of T_m.

(2) *The ℓ_1 norm of the diagonal of the largest tile tends to zero as $k \to \infty$.*

Proof Let T be an arbitrary tile defined by $T = \{x : a \leq x \leq b\}$. T is subdivided into two subtiles using a hyperplane perpendicular to T's longest edge. The length of this edge is at least $\|T\|_1/n$ before subdivision, and at most $A\|T\|_1/(n + An)$ afterwards. Hence, subdivision scales $\|T\|_1$ by a factor that is at most $1 - 1/(n + An)$, giving the first result.

For the second result, we note that there are at most 2^m tiles that are generated by subdividing Ω m times. At each iteration the largest tile is always subdivided. Hence after $k = 2^{m+1} - 1$ iterations there are no tiles with 1-norm larger than

$$\|T_m\|_1 \leq \left(1 - \frac{1}{(1+A)n}\right)^m \|\Omega\|_1. \tag{3}$$

Clearly as $k \to \infty$, m also goes to infinity, and the maximum tile 1-norm goes to zero. ■

In the case when each subdivision cuts a tile into two equally sized halves, the bound in (3) is at its least, which is

$$\|T_m\|_1 \leq \left(1 - \frac{1}{2n}\right)^m \|\Omega\|_1.$$

Proposition 3.3 allows us to show that without restarts, the algorithm will find a point at least as good as the open set essential minimum.

COROLLARY 3.4 *Without restarts, the limit* f_∞ *of the sequence of best function values* $\{f^{(k)}\}$ *satisfies* $f_\infty \leq f^\sharp$.

Proof Let x be an arbitrary point in Ω. At iteration k the point x lies in some tile T of the current set of tiles. The 1-norm of T is bounded above by

$$\|T\|_1 \leq \left(1 - \frac{1}{(1+A)n}\right)^{\lfloor \log_2(k+1) \rfloor - 1} \|\Omega\|_1, \tag{4}$$

where the floor function $\lfloor s \rfloor$ is the greatest integer less than or equal to s. Hence there is at least one iterate within the distance (4) of x. As $k \to \infty$ the bound in (4) goes to zero, and so x is a cluster point of the sequence of iterates. Since $x \in \Omega$ was arbitrary, it follows that the sequence of sample points is dense in Ω.

For any $f_0 > f^\sharp$ there exists an open ball $B \subset \Omega$ such that $f(x) < f_0$ for all $x \in B$. The method will place a sample point inside B within a finite time, meaning $f_\infty \leq f_0$ for all $f_0 > f^\sharp$. Hence $f_\infty \leq f^\sharp$. ■

Convergence with probability 1 to the essential global minimum stems from an intermediate result, which is given next.

PROPOSITION 3.5 *The solution to the problem of maximizing*

$$\Xi = \prod_{i=1}^{N} (1 - \gamma_i) \quad \textit{subject to} \quad \sum_{i=1}^{N} \gamma_i = \Gamma \quad \textit{and} \quad \gamma_i \in [0, 1] \quad \forall i$$

with $0 < \Gamma < N$ *is* $\gamma_i = \Gamma/N$ *for all* $i = 1, \ldots, N$.

Proof Ξ is zero if $\gamma_i = 1$ for any i, whereas it is positive if $\gamma_i = \Gamma/N$ for all i, so the upper bounds are redundant. Also the maximizer does not have any $\gamma_i = 0$. To see this choose $\gamma_i = 0$ and $\gamma_j > 0$. The latter choice is guaranteed for some j because $\Gamma > 0$. Let $(1 - \gamma_i) + (1 - \gamma_j) = B$ say, where $B \in (1, 2)$. Simple calculations show that Ξ is maximized with respect to γ_i and γ_j when $1 - \gamma_i = 1 - \gamma_j = B/2$, implying neither γ_i nor γ_j is zero.

Hence, we can find the global maximum of Ξ by examining the critical points of Ξ subject to the equality constraint $\sum_{i=1}^{N} \gamma_i = \Gamma$. The Karush–Kuhn–Tucker conditions for this problem are

$$-\frac{\Xi}{1 - \gamma_i} = \lambda \quad \forall i = 1, \ldots, N \quad \text{and} \quad \sum_{i=1}^{N} \gamma_i = \Gamma,$$

where λ is the Lagrange multiplier of the constraint. The first of these equations shows that all γ_i take the same value. The second shows that this value must be Γ/N, as required. ■

The next theorem states the second asymptotic result for TILECUTTER. This result shows convergence in probability to the essential global minimum with or without restarts. It makes use of the following class of functions.

DEFINITION 3.6 *The class $\mathcal{F}(f)$ is the set of Lebesgue measurable functions with the same marginal distributions as f. That is to say $g(x) \in \mathcal{F}(f)$ implies*

$$\frac{1}{m(\Omega)} \int_{\Omega} U(f(x) - \phi)\, \mathrm{d}x = \frac{1}{m(\Omega)} \int_{\Omega} U(g(x) - \phi)\, \mathrm{d}x \quad \forall \phi \in R,$$

where $U(x)$ is the step function given by $U(x) = 1$ if $x > 0$, and $U(x) = 0$ otherwise. Here $m(\cdot)$ is the Lebesgue measure.

THEOREM 3.7 *The probability that the best-known function value $f^{(k)}$ at iteration k exceeds a specified level $\phi > f^*$ is given by*

$$P(f^{(k)} > \phi) \leq \left(1 - \frac{m(L(\phi))}{m(\Omega)}\right)^{\sum N_i}, \tag{5}$$

where N_i is the number of tiles cut from Ω between the ith and $(i+1)$th restarts in the first k iterations. Here, $L(\phi)$ is the level set $\{x \in \Omega : f(x) \leq \phi\}$.

Proof Initially, we examine the case when TILECUTTER has generated N function values in tiles cut from Ω, without restarts. The function value of the lowest of these N points is labelled f_N. We seek an upper bound on the worst case probability of failure $P(f_N > \phi)$, where $\phi > f^*$ is the value separating success and failure. Let $\mathcal{F}(f)$ be the set of Lebesgue measurable functions on Ω with the same marginal distributions as f. We seek the upper bound

$$P(f_N > \phi) \leq \sup_{g \in \mathcal{F}(f), \mathcal{T}_N} P(f_N > \phi | \text{ tiling strategy } \mathcal{T}_N \text{ and objective function } g),$$

where the supremum is over all possible functions in $g \in \mathcal{F}(f)$ and all possible tilings of Ω consisting of N tiles.

The condition $\phi > f^*$ means level set $L(\phi) = \{x \in \Omega : f(x) \leq \phi\}$ has a strictly positive measure $\mu m(\Omega)$. The arbitrary tiling strategy $\mathcal{T}_N$ divides Ω into N tiles with measures $\tau_1, \ldots, \tau_N$. Let μ_i be the measure of the intersection of $L(\phi)$ with tile i. Each of these tiles contains one random sample point, and so the probability of no sample points lying in $L(\phi)$ with this tiling is

$$P(f_N > \phi) = \prod_{i=1}^{N} \left(1 - \frac{\mu_i}{\tau_i}\right).$$

Proposition 3.5 shows that this is maximized when $\mu_i = \tau_i \mu$ for all i. Hence

$$P(f_N > \phi) \leq \left(1 - \frac{m(L(\phi))}{m(\Omega)}\right)^{N}. \tag{6}$$

Without restarts, $N \geq k$ and so $P(f_N > \phi) \to 0$ and hence $P(f^{(k)} > \phi) \to 0$ as $k \to \infty$.

With restarts the algorithm does a series of batches of points of sizes $N_1, N_2, \ldots$, where N_i is the number of points placed in tiles cut from Ω as the initial tile between the ith and $(i+1)$th restarts. Here the initial start counts as the first restart. Since the largest tiles are cut first, the first point in each batch is placed in Ω, which means $N_i \geq 1$ for all i. Formula (6) gives an upper bound on the probability that the ith restart will not locate a point at least as low as ϕ. This formula only considers the chance of success from subdividing Ω, and so the bounds for each restart are independent of one another. Hence

$$P(f^{(k)} > \phi) \leq \left(1 - \frac{m(L(\phi))}{m(\Omega)}\right)^{\sum N_i}.$$

Now $\sum N_i \to \infty$ as $k \to \infty$, and so $P(f^{(k)} > \phi) \to 0$ as $k \to \infty$, as required. ■

At first sight the bound in (5) appears pessimistic. Actually it is tight. To see this, we define a sequence of functions $\{g_j(x)\}$ on Ω via

$$g_j(x) = f\left(\ell + \sum_{i=1}^{n}[j(x_i - \ell_i) \mod (u_i - \ell_i)]\underline{e}_i\right),$$

where $\underline{e}_i$ is the ith column of the identity matrix. Essentially g_j covers Ω with j^n copies of f on Ω, where each copy has been shrunk by a factor j along every direction, and the copies are laid out in a $j \times j \times \cdots \times j$ grid. The logic behind this is that any tile T consisting of whole copies of f has the same marginal distribution for g_j that f has on Ω. For tiles that do not consist of whole copies of f, the difference in the two marginal distributions is at most the relative volume of the copies of f that touch the boundary of T. Let T be defined by the bounds $a_i \le x_i \le b_i$ and Ω by $\ell_i \le x_i \le u_i$, both for $i = 1, \ldots, n$. Then the volume of the copies of f intersecting the boundary of T is at most

$$m(T) - \prod_{i=1}^{n}\left[(b_i - a_i) - 2\frac{u_i - \ell_i}{j}\right].$$

Hence the difference in the marginal distributions of f on Ω and g_j on T is bounded by

$$\left|\frac{1}{m(\Omega)}\int_{\Omega} U(f(x) - \phi)\,\mathrm{d}x - \frac{1}{m(T)}\int_{T} U(g_j(x) - \phi)\,\mathrm{d}x\right| \le 1 - \frac{1}{m(T)}\left(\prod_{i=1}^{n}\left[(b_i - a_i) - 2\frac{u_i - \ell_i}{j}\right]\right).$$

As j increases it is clear that the ratio $m(\{x \in T : g_j(x) \le \phi\})/m(T)$ approaches that of $m(L(\phi))/m(\Omega)$ for any fixed tile T, which is the condition needed to achieve the worst case result in the proof of Theorem 3.7.

COROLLARY 3.8 *The limit f_∞ of the sequence of best function values $\{f^{(k)}\}$ satisfies $f_\infty \le f^*$ with probability* 1 *irrespective of whether or not restarts are used.*

Proof Choose $\phi > f^*$. The definition of the essential global minimum f^* means that $m(L(\phi))$ is strictly positive. Hence Theorem 3.7 shows that the probability that no sample point lies in $L(\phi)$ goes to zero as the number sample points placed in tiles cut from Ω goes to infinity. ■

If all restarts use Ω as the sole initial tile (i.e. no extra initial tiles are used) then $P(f_N > \phi)$ is independent of the number of restarts and the number of function evaluations between restarts. Equation (6) shows that, at worst, the expected performance of the algorithm is the same as PRS.

COROLLARY 3.9 *If f is continuous on Ω at every point in some level set $L(\psi)$ with $\psi > f^*$, then $f(x_\infty) = \min\{f(x) : x \in \Omega\}$ almost surely, where x_∞ is an arbitrary cluster point of the sequence of best points $\{x^{(k)}\}$.*

Proof Corollary 3.8 shows $f_\infty \le f^*$ almost surely. Hence $x^{(k)} \in L(\psi)$ for all k sufficiently large. $L(\psi)$ is a closed and bounded subset of R^n and thus is compact. This implies $x_\infty \in L(\psi)$. Continuity at x_∞ implies $f(x_\infty) = f_\infty \le f^*$. Finally continuity at all points in $L(\psi)$ implies that the set of points in Ω satisfying $f < f^*$ must be empty. Hence $f(x_\infty) = f^*$ almost surely. Standard analysis techniques can then be used to show $f^* = \min\{f(x) : x \in \Omega\}$, giving the result. ■

4. Numerical results and discussion

Numerical results were generated for two sets of test functions with and without restarts. The first set is that used by Rinnooy Kan and Timmer [15]. The second is chosen from the larger set of test functions presented in Ali *et al.* [2], together with one function (trigonometric) from [11]. The former were used as smooth functions as they are not easily made non-smooth. The ones chosen from the latter set are sums of squares, and can easily be made non-smooth by replacing the squares with absolute values.

Numerical results were generated using $A = \frac{3}{2}$ and $\tau_{\text{acc}} = 10^{-8}$. The algorithm is relatively insensitive to these parameters, with the exception of A when $A = 1$, as discussed in Section 2.2.

Numerical results for test set 1 [15] without restarts are listed in Table 1. These show TILECUTTER was able to locate the global minima to high accuracy for all but the Shekel problems. For the latter, TILECUTTER located the basin of attraction of the global minimizer, but was unable to refine its estimate of the global minimizer to high accuracy within 1600 function evaluations. The authors also applied the clustering strategy of Rinnooy Kan *et al.* [14,15] to the sample points generated by TILECUTTER. This showed our method located the basin of the global minimizer as quickly as the multi-level single linkage algorithm of [14,15]. The latter then performs a local search from the lowest point in each cluster using a Newton method, whereas TILECUTTER refines its estimate of the global minimizer by continuing the global search strategy.

Seven of the test set 1 problems have results reported in [9] for the C-GRASP method. C-GRASP is a stochastic multistart algorithm. Its local search uses a stochastic local exploration strategy in conjunction with a line search. Hirsch *et al.* [9] list the average number of function evaluations taken by C-GRASP to satisfy

$$|f^{(k)} - f^*| < 10^{-6} + 10^{-4}|f^*|,$$

where f^* is the global minimum. Using the same stopping condition the averages for TILECUTTER and C-GRASP (in brackets) are: Branin, 717 (59,857); Goldstein-Price, 771 (29); Shekel5, 5449 (5,545,982); Shekel7, 4475 (4,052,800); Shekel10, 6295 (4,701,358); Hartmann3, 1205 (20,743); and Hartmann6, 12,504 (79,685). On all but one of these problems TILECUTTER is clearly superior. On Goldstein–Price TILECUTTER is still effective, but C-GRASP is exceptionally quick.

Numerical results were also generated *with* restarts for functions in test set 1. These are listed in Table 2. For each function the average lowest function value found by the algorithm after 1600 function evaluations is listed, where each average is over 10 runs. Results are listed for 1, 50, 200, and 800 function evaluations between restarts, and also for no restarts. For the first of these values

Table 1. Rate of progress on Rinnooy Kan and Timmer's test problems.

			Number of function evaluations					
Function	n	f^*	50	100	200	400	800	1600
Brannin	2	0.3979	0.657	0.436	0.406	0.3989	0.3979	0.3979
Goldstein	2	3.0000	10.1	4.41	3.11	3.01	3.0006	3.0000
Camel 6	2	−1.0316	−0.73	−0.95	−1.01	−1.03	−1.0315	−1.0316
Shekel 5	4	−10.1532	−0.73	−1.26	−2.59	−4.81	−7.29	−9.85
Shekel 7	4	−10.4029	−0.90	−1.64	−2.40	−4.41	−8.12	−10.0
Shekel 10	4	−10.5364	−2.09	−2.70	−3.13	−4.90	−7.62	−10.0
Hartmann 3	3	−3.8628	−3.67	−3.79	−3.84	−3.858	−3.862	−3.8624
Hartmann 6	6	−3.3224	−2.26	−2.49	−2.83	−3.01	−3.18	−3.26
Rastrigan	2	−2	−1.62	−1.77	−1.89	−1.991	−2	−2

The first three columns list the problem name, dimension, and global minimum. The next six columns list the average function value (over 10 runs) found by tilecutter after 50, 100, ..., 1600 function evaluations. No restarts and $A = \frac{3}{2}$ were used.

Table 2. The effect of restarts on Rinnooy Kan and Timmer's test problems.

			Function evaluations per restart				
Function	n	f^*	1	50	200	800	∞
Brannin	2	0.3979	0.42	0.409	0.399	0.3979	0.3979
Goldstein	2	3.0000	3.74	3.34	3.01	3.0001	3.0000
Camel 6	2	−1.0316	−0.98	−1.02	−1.03	−1.0316	−1.0316
Shekel 5	4	−10.1532	−1.76	−1.67	−3.87	−8.31	−9.85
Shekel 7	4	−10.4029	−1.75	−3.03	−3.57	−8.20	−10.0
Shekel 10	4	−10.5364	−2.29	−2.59	−3.91	−8.64	−10.0
Hartmann 3	3	−3.8628	−3.84	−3.84	−3.858	−3.8621	−3.8624
Hartmann 6	6	−3.3224	−2.76	−2.91	−3.02	−3.1932	−3.26
Rastrigan	2	−2	−1.92	−1.93	−1.985	−2	−2

The first three columns list the problem name, dimension, and global minimum. The next five columns list the average function value (over 10 runs) found by TILECUTTER after 1600 function evaluations, with restarts after every 1, 50, 200, or 800 function evaluations, and without restarts, respectively ($A = \frac{3}{2}$).

TILECUTTER degenerates to PRS. Table 2 shows that the performance of the TILECUTTER degrades steadily as the number of function evaluations between restarts declines. Interestingly, there is little difference between results for PRS and TILECUTTER with 50 evaluations between restarts. The opinion of the author is that 50 function evaluations is insufficient for TILECUTTER to identify and exploit promising regions. Theorem 3.7 shows the *worst* case performance of the method is equivalent to that of PRS. The functions in test set 1 are not 'worst case' and so TILECUTTER is able to make meaningful gains over PRS when restarts are not too frequent.

4.1 *Non-smooth test problems*

Functions were selected from the test set in [2] with the property that they were sums of squares $\sum_i f_i^2$ with a global minimum of zero. These were made into non-smooth functions of the form $f = \sum_i |f_i|$. The fact that $f_i = 0$ for all i at the global minimizer means both forms of the test function have the same global minimum (zero) and global minimizer(s). Numerical results with $A = \frac{3}{2}$ are presented in Table 3. Replacing the squares with absolute values makes the function values obtained look deceptively poor. For example, a final function value of 10^{-5} on the modified function corresponds to a final value of about 10^{-10} on the original form of the problem. The results show reasonably steady decreases in objective function value, indicating that the method located the general areas containing the global minimizers early on.

Table 3. Rate of progress on the non-smooth versions of Ali's test problems [2].

		Number of function evaluations						
Function	n	200	400	800	1600	3200	6400	12800
Becker and Lago	2	0.07	0.014	0.002	$3e{-4}$	$1e{-5}$	$2e{-6}$	$3e{-8}$
Bohachevsky 1	2	1.01	0.646	0.491	0.359	0.255	0.067	$2e{-3}$
Bohachevsky 2	2	0.66	0.27	0.17	0.13	0.10	0.009	$9e{-5}$
Levy and Montalvo 1	3	1.10	0.56	0.19	0.043	0.011	0.002	$3e{-4}$
Levy and Montalvo 2	3	0.12	0.068	0.027	0.006	0.002	$5e{-4}$	$8e{-5}$
Mod. Rosenbrock	2	0.30	0.13	0.074	0.014	$4e{-3}$	$2e{-4}$	$2e{-6}$
Periodic (Price)	2	0.15	0.09	0.067	0.063	0.019	0.004	$2e{-7}$
Schaffer 2	2	1.6	0.85	0.38	0.13	0.028	0.004	$6e{-4}$
Trigonometric	5	0.12	0.089	0.053	0.024	0.011	0.005	0.001

The first two columns list the problem name and dimension. The next eight columns list the average function value (over 10 runs) found by tilecutter after 200, ..., 12800 function evaluations. In each case, the optimal function value is zero. No restarts and $A = \frac{3}{2}$ were used.

Table 4. Rate of progress on the non-smooth versions of Ali's test problems [2].

		Number of function evaluations						
Function	n	200	400	800	1600	3200	6400	12800
Becker and Lago	2	0.006	6e−4	9e−6	1e−7	6e−10	7e−14	0
Bohachevsky 1	2	0.669	0.397	0.312	0.205	0.167	0.013	0.004
Bohachevsky 2	2	0.236	0.173	0.166	0.130	0.029	2e−7	2e−15
Levy and Montalvo 1	3	0.495	0.222	0.036	0.006	0.004	0.003	3e−4
Levy and Montalvo 2	3	0.037	0.012	0.004	4e−4	9e−5	4e−5	5e−7
Mod. Rosenbrock	2	0.067	0.025	0.008	0.002	1e−7	2e−12	3e−15
Periodic (Price)	2	0.108	0.091	0.082	0.079	0.040	0.005	0.001
Schaffer 2	2	0.561	0.094	0.012	0.001	3e−5	2e−7	1e−10
Trigonometric	5	0.085	0.046	0.024	0.008	0.003	3e−4	1e−4

The first two columns list the problem name and dimension. The next eight columns list the average function value (over 10 runs) found by TILECUTTER after 200, ..., 12800 function evaluations. In each case the optimal function value is zero. No restarts and $A = 1$ were used.

Table 5. Rate of progress of the accelerated random search algorithm on the non-smooth versions of Ali's test problems [2].

		Number of function evaluations						
Function	n	200	400	800	1600	3200	6400	12800
Becker and Lago	2	0.011	8e−4	4e−5	3e−5	2e−5	1e−5	7e−6
Bohachevsky 1	2	0.590	0.474	0.387	0.308	0.287	0.245	0.038
Bohachevsky 2	2	0.192	0.135	0.131	0.113	0.082	0.021	1e−3
Levy and Montalvo 1	3	1.156	1.077	0.546	0.490	0.344	0.007	3e−5
Levy and Montalvo 2	3	0.055	0.031	0.024	0.021	0.017	0.007	8e−6
Mod. Rosenbrock	2	0.147	0.059	0.054	0.053	0.053	0.052	0.052
Periodic (Price)	2	0.107	0.100	0.100	0.100	0.090	0.080	0.080
Schaffer 2	2	0.509	0.118	0.025	0.018	0.016	0.013	0.012
Trigonometric	5	0.057	0.042	0.035	0.032	0.027	0.020	0.010

The first two columns list the problem name and dimension. The next eight columns list the average function value (over 10 runs) found by TILECUTTER after 200, ..., 12, 800 function evaluations. In each case the optimal function value is zero.

The algorithm was compared with ARS and to itself with $A = 1$ on test set 2, with results listed in Tables 3–5. The comparison between the $A = \frac{3}{2}$ and 1 cases shows that the latter is better on seven problems, and equally good on another. The stronger focus on searching locally with $A = 1$ is clear, and this is effective on test set 2.

A comparison between TILECUTTER and ARS (Tables 3 and 5) showed that TILECUTTER was faster on most problems. ARS appears to have stalled on modified Rosenbrock's, and has become trapped in a local minimizer on the periodic (Price) function.

Comparisons with a version of DIRECT were also made. This version cut every Pareto optimal tile with size not less than τ_{acc} at each iteration. Four of these test functions were omitted because their global minimizers lie at the centrepoint of Ω, which is the first point DIRECT calculates f at. Results for DIRECT on the remaining problems are listed in Table 6. These five have their solutions near the centre of Ω. The search strategy of DIRECT is essentially from the centre outwards, and it outperforms TILECUTTER on these problems. The more local character of the search strategy of DIRECT is clear at higher function counts. TILECUTTER puts much more effort into searching globally, with the consequence that it refines the best known minimizer much more slowly.

The method was also compared with DIRECT and ARS on a small number of new non-smooth problems. The first two are

$$f_{\text{weka 1}} = \min\left(1 + \sqrt{x_1^2 + x_2^2}, 4x_1 + 4\right) \quad \text{over } x \in [-1, 1]^2$$

Table 6. Rate of progress on the non-smooth versions of Ali's test problems [2].

		Number of function evaluations						
Function	n	200	400	800	1600	3200	6400	12800
Becker and Lago	2	0.12	0.014	0.002	5e−5	2e−9	2e−9	2e−9
Levy and Montalvo 1	3	0.04	2e−3	2e−4	2e−6	1e−9	1e−9	1e−9
Levy and Montalvo 2	3	0.004	4e−4	2e−5	3e−7	4e−10	4e−10	4e−10
Mod. Rosenbrock	2	0.17	0.04	2e−3	4e−7	4e−9	4e−9	4e−9
Trigonometric	5	0.10	0.048	0.02	0.01	4e−4	5e−7	2e−9

The first two columns list the problem name and dimension. The next eight columns list the function value found by DIRECT after 200, ..., 12800 function evaluations. In each case the optimal function value is zero.

and

$$f_{\text{weka 2}} = \sum_{i=1}^{n} (k_i \lfloor p_i x_i \rfloor \mod p_i) + \sum_{i=1}^{n} x_i(1 - x_i) \quad \text{over } x \in [0, 1]^n.$$

Here k_i and p_i are positive integers, the p_i are mutually co-prime, and $k_i < p_i$ for all i. Numerical results were generated using $n = 2$, $k_1 = 11$, $k_2 = 12$, $p_1 = 17$, and $p_2 = 19$. The third new problem is

$$f_{\text{weka 3}} = \sum_{j=0}^{\infty} \left[3^{(-j)} \sum_{i=1}^{n} (k_i \lfloor 3^j p_i x_i \rfloor \mod p_i) \right] + \sum_{i=1}^{n} x_i(1 - x_i)$$

also with $\Omega = [0, 1]^n$. This function was implemented in double precision by terminating the outer sum at $j = 40$ because any remaining terms are well-below machine precision in comparison to the leading terms.

Results for these three problems appear in Table 7. These show that TILECUTTER and DIRECT are both effective, with the former being faster. ARS focused more on searching locally rather than globally. This local focus means ARS outperforms the others on weka 1 (with only two minima) but does poorly on the weka 2 and 3 where many local minima are present.

The final additional problem is

$$f_{\text{weka 4}} = \min(2 + x_1 \cos(\theta) + x_2 \sin(\theta), 40\|x - c\|_2) \quad \text{over } x \in [-1, 1]^2,$$

Table 7. Rate of progress on the first three weka test problems.

		Number of function evaluations					
Function	Method	50	100	200	400	800	1600
weka 1	ARS	0.056	0.002	2e−4	3e−6	1e−6	3e−7
weka 1	TILECUTTER	0.05	0.017	0.003	0.002	8e−4	6e−5
weka 1	DIRECT	0.148	0.048	0.048	0.016	0.016	0.005
weka 2	ARS	2.3	1.57	1.25	1.25	1.23	1.12
weka 2	TILECUTTER	2.27	1.4	0.49	0.12	0.11	5e−5
weka 2	DIRECT	4.2	1.25	1.24	1.24	1.24	4e−4
weka 3	ARS	7.6	6.4	5.5	5.1	4.1	3.1
weka 3	TILECUTTER	7.8	4.5	2.01	0.49	0.026	0.002
weka 3	DIRECT	11	4.71	2.83	2.76	2.76	0.03

The first two columns list the problem number and method. For each problem $f^* = 0$ and $n = 2$. Average function values (over 10 runs) found by ARS and tilecutter (without restarts, and with $A = \frac{3}{2}$) after 50, 100, ..., 1600 function evaluations. The rows for DIRECT list the function value after the relevant number of function evaluations.

Table 8. Time taken (in seconds) to execute DIRECT and TILECUTTER on the Zakharov problem in 10 dimensions.

	Number of function evaluations								
	100	300	1×10^3	3×10^3	1×10^4	3×10^4	1×10^5	3×10^5	1×10^6
Function	0.008	0.034	0.075	0.235	0.76	2.43	8.19	24	78
DIRECT	0.033	0.070	0.186	0.593	2.23	12.24	46.39	235	1757
TILECUTTER	0.043	0.073	0.204	0.495	1.71	5.44	19.34	74	389

The first row lists the number of function evaluations performed. The second row lists the time taken for a for loop to do this number of function evaluations. The last two rows list the times taken by the two algorithms. No restarts and $A = \frac{3}{2}$ were used.

where $\theta \in [0, 2\pi]$ and $c \in [-1, 1]^2$ are chosen randomly. The bulk of this function is determined by the sloping plane, but the global minimizer is determined by the randomly sited cone. In contrast to most of the other test problems, most of this function provides no information about where the global mimimizer lies. One thousand different instances of this problem were generated, and solved by both DIRECT and TILECUTTER. DIRECT and TILECUTTER took an average of 9463 and 8028 function evaluation, respectively, with the standard deviation in the difference of the means being 267. Here TILECUTTER is faster than DIRECT by a margin greater than five standard deviations.

4.2 *Overheads*

The costs of the overheads for DIRECT and TILECUTTER are explored in Table 8 using the Zakharov test function in 10 dimensions. Each algorithm was halted after the number of function evaluations listed in the first row. The second row lists the time needed to evaluate the function that many times inside a for loop. The time taken by the two algorithms (including function evaluations) are listed in the final two rows. All test runs were performed on a multi-user system with other users present. Up to 1000 function evaluations DIRECT is slightly quicker, with TILECUTTER faster from 3000 function evaluations up. For DIRECT the time spent on overheads per function evaluation rises from 110 microseconds (μs) for 1000 function evaluations, to 1700 μs at 10^6 function evaluations. In contrast, the rise is much less dramatic for TILECUTTER: from 130 to 311 μs. The reason for this is simple: DIRECT typically cuts up fewer tiles per iteration than TILECUTTER does when $A > 1$, as described in Section 2.2. Hence DIRECT has to perform the computationally expensive process of finding which tiles are Pareto optimal many more times for the same number of function evaluations.

5. Conclusion

A cover partitioning method for global optimization of non-smooth and discontinuous functions subject to simple bounds is presented. The method is stochastic and so a limited memory version can be constructed by restarting the method from time to time. It is shown that both the full and limited memory versions converge to the essential global minimum with probability 1. In addition, without restarts the algorithm is guaranteed to find the open set essential global minimum. Numerical results show that TILECUTTER is effective on smooth and non-smooth functions. Numerical experiments show that infrequent restarting has little impact on performance, but performance degrades if restarting becomes more frequent.

The algorithm is compared with DIRECT, ARS, and C-GRASP. It outperforms the latter two on the test functions used, and is comparable to the former. TILECUTTER and DIRECT have different strengths. TILECUTTER weights searching globally more heavily than DIRECT, at the expense of refinement of the best known minimizer(s). These differences in strategy were reflected in the numerical results.

Acknowledgements

The authors would like to thank both referees whose insightful comments have led to a much improved form of this paper.

References

[1] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas, *A global optimization method, αBB, for general twice differentiable constrained NLPs — II. Implementation and computational results*, Comput. Chem. Eng. 22 (1998), pp. 1159–1179.
[2] M.M. Ali, C. Khompatraporn, and Z.B. Zabinsky, *A numerical evaluation of several stochastic algorithms on selected continuous global optimization problems*, J. Global Optim. 31 (2005), pp. 635–672.
[3] R. Andreani, E.G. Birgin, J.M. Martínez, and M.L. Schuverdt, *On augmented Lagrangian methods with general lower level constraints*, SIAM J. Optim. 18 (2007), pp. 1286–1309.
[4] R. Andreani, E.G. Birgin, J.M. Martínez, and M.L. Schuverdt, *Augmented Lagrangian methods under the constant positive linear dependence constraint qualification*, Math. Prog. Ser. B 111 (2008), pp. 5–32.
[5] M.J. Appel, R. Labarre, and D. Radulović, *On accelerated random search*, SIAM J. Optim. 14 (2003), pp. 708–731.
[6] C. Audet and J.E. Dennis Jr., *Analysis of generalized pattern searches*, SIAM J. Optim. 13 (2003), pp. 889–903.
[7] E.G. Birgin, C.A. Floudas, and J.M. Martínez, *Global minimization using an augmented Lagrangian method with variable lower-level constraints*, Math. Prog. Ser. A 125 (2010), pp. 139–162.
[8] R. Hettich and K.O. Kortanek, *Semi-infinite programming: theory, methods, and applications*, SIAM Rev. 35 (1993), pp. 380–429.
[9] M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende, *Global optimization by continuous GRASP*, Optim. Lett. 1 (2007), pp. 201–212.
[10] D. Jones, C.D. Perttunen, and B.E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, J. Optim. Theory Appl. 79 (1993), pp. 157–181.
[11] J.J. Moré, B.S. Garbow, and K.E. Hillstrom, *Testing unconstrained optimization software*, ACM Trans. Math. Softw. 7 (1981), pp. 17–41.
[12] J. Pinter, *Convergence qualification of adaptive partition algorithms in global optimization*, Math. Prog. 56 (1992), pp. 343–360.
[13] L. Pronzanto, E. Walter, A Venot, and J.-F. Lebruchec, *A general purpose global optimizer: implementation and applications*, Math. Comput. Simul. XXVI (1984), pp. 412–422.
[14] A.H.G. Rinnooy Kan and G.T. Timmer, *Stochastic global optimization methods part I: clustering methods*, Math. Prog. 39 (1987), pp. 27–56.
[15] A.H.G. Rinnooy Kan and G.T. Timmer, *Stochastic global optimization methods part II: multi-level methods*, Math. Prog. 39 (1987), pp. 57–78.
[16] Z.Bo. Tang, *Adaptive partitioned random search to global optimization*, IEEE Trans. Auto. Control 11 (1994), pp. 2235–2244.
[17] A. Törn and A. Žilinskas, *Global Optimization*, Lecture Notes in Computer Science, vol. 350, Springer-Verlag, Berlin/Heidelberg/New York, 1989.
[18] T. Ye and S. Kalyanaraman, *A recursive random search algorithm for large scale network parameter configuration*, Proceedings of the ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems, San Diego, 9–14 June 2003, pp. 196–205.