

*Branching and bounding improvements
for global optimization algorithms with
Lipschitz continuity properties*

**Coralia Cartis, Jaroslav M. Fowkes &
Nicholas I. M. Gould**

Journal of Global Optimization

An International Journal Dealing with
Theoretical and Computational Aspects
of Seeking Global Optima and Their
Applications in Science, Management
and Engineering

ISSN 0925-5001

Volume 61

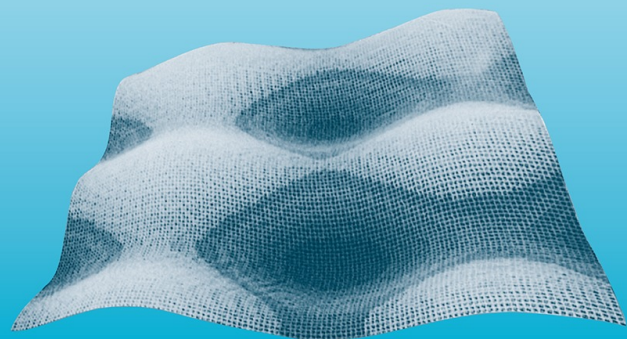
Number 3

J Glob Optim (2015) 61:429-457
DOI 10.1007/s10898-014-0199-6

Volume 61, No. 3 – March 2015

ISSN 0925-5001

**Journal
of Global
Optimization**



 Springer

 Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Branching and bounding improvements for global optimization algorithms with Lipschitz continuity properties

Coralia Cartis · Jaroslav M. Fowkes ·
Nicholas I. M. Gould

Received: 5 June 2013 / Accepted: 2 May 2014 / Published online: 15 May 2014
© Springer Science+Business Media New York 2014

Abstract We present improvements to branch and bound techniques for globally optimizing functions with Lipschitz continuity properties by developing novel bounding procedures and parallelisation strategies. The bounding procedures involve nonconvex quadratic or cubic lower bounds on the objective and use estimates of the spectrum of the Hessian or derivative tensor, respectively. As the nonconvex lower bounds are only tractable if solved over Euclidean balls, we implement them in the context of a recent branch and bound algorithm (Fowkes et al. in J Glob Optim 56:1791–1815, 2013) that uses overlapping balls. Compared to the rectangular tessellations of traditional branch and bound, overlapping ball coverings result in an increased number of subproblems that need to be solved and hence makes the need for their parallelization even more stringent and challenging. We develop parallel variants based on both data- and task-parallel paradigms, which we test on an HPC cluster on standard test problems with promising results.

Keywords Global optimization · Lipschitzian optimization · Parallel branch and bound · Nonconvex programming

1 Introduction

In many applications one encounters the global optimization problem

$$\min_{x \in \mathcal{D}} f(x), \quad (1.1)$$

C. Cartis · J. M. Fowkes (✉)
School of Mathematics, University of Edinburgh, The King's Buildings,
Edinburgh EH9 3JZ, Scotland, UK
e-mail: jaroslav.fowkes@ed.ac.uk

C. Cartis
e-mail: coralia.cartis@ed.ac.uk

N. I. M. Gould
Computational Science and Engineering Department, Rutherford Appleton Laboratory,
Chilton, Oxfordshire OX11 0QX, UK
e-mail: nick.gould@stfc.ac.uk

where $f : \mathcal{D} \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is smooth and in general non-convex and \mathcal{D} is a compact, convex set. It has been shown that this problem is NP-hard [23] and requires global information to be solved efficiently [39]. Branch and bound algorithms are a traditional way to solve (1.1) (see for example [20, 29, 32]). Such algorithms work by recursively splitting (branching) the domain \mathcal{D} into subregions and bounding the objective function f over each subregion until the global minimum is found.

In order for such algorithms to be efficient, one requires accurate and efficiently computable lower bounds on f over each subregion (upper bounds are typically taken to be the function evaluated at some point or the outcome of a local solver). Global information in the form of a Lipschitz constant is often used to construct such lower bounds. The case where the lower bound is based on a Lipschitz constant of the objective function f has the immediate form $f(x) \geq f(x_{\mathcal{B}}) - L_f(\mathcal{B})\|x - x_{\mathcal{B}}\|$ for some point $x_{\mathcal{B}}$ in a subregion \mathcal{B} and a Lipschitz constant $L_f(\mathcal{B})$ for f over \mathcal{B} ; this case has been well studied in the global optimization literature (see [10, 26, 29, 30, 32, 33, 36, 38, 40] and references therein).

A more accurate lower bound using a Lipschitz constant of the gradient of the objective function $g = \nabla_x f$ can be derived using Taylor's theorem to first order

$$f(x) \geq q_{\mathcal{B}}(x) := f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) - \frac{L_g(\mathcal{B})}{2} \|x - x_{\mathcal{B}}\|_2^2 \tag{1.2}$$

for some point $x_{\mathcal{B}}$ in a subregion \mathcal{B} and the gradient's Lipschitz constant $L_g(\mathcal{B})$ for g over \mathcal{B} , and has, together with refinements, also been well studied (see [3, 4, 11, 12, 16, 24, 25, 27, 35]). Evtushenko [10], Baritompa and Cutler [3], Evtushenko and Posypkin [11, 12] have replaced $-L_g(\mathcal{B})$ in (1.2) with a lower bound on the spectrum of the Hessian. Baritompa and Cutler [3] also go a step further and replace the simple quadratic term in (1.2) by a quadratic form that lower bounds f .

The case where one goes even further and uses second order Taylor's theorem to obtain a cubic lower bound using a Lipschitz constant for the Hessian $H = \nabla_{xx} f$ was considered in [16]

$$f(x) \geq c_{\mathcal{B}}(x) := f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) - \frac{L_H(\mathcal{B})}{6} \|x - x_{\mathcal{B}}\|_2^3 \tag{1.3}$$

for some point $x_{\mathcal{B}}$ in a subregion \mathcal{B} and the Hessian's Lipschitz constant $L_H(\mathcal{B})$ for H over \mathcal{B} .

In this paper, we propose new bounding techniques using refinements of the bounds (1.2) and (1.3). We show that for the first order bound (1.2) it is possible to obtain tighter results by using existing lower bounds on the spectrum of the Hessian, some of which have not been previously used, as far as we are aware, in the context of Lipschitz based global optimization. Additionally, we extend one of these approaches to the second order bound (1.3) by replacing the Lipschitz constant estimate with a novel lower bound on the spectrum of the third order derivative tensor. We test the new proposals in the Overlapping Branch and Bound (oBB) framework proposed in [16] which allows efficient global solution of the non-convex lower bounding subproblems

$$\min_{x \in \mathcal{B}} l_{\mathcal{B}}(x) \tag{1.4}$$

where $l_{\mathcal{B}}(x)$ is either $q_{\mathcal{B}}(x)$ in (1.2) or $c_{\mathcal{B}}(x)$ in (1.3), over each subdomain \mathcal{B} , by letting \mathcal{B} be a Euclidean ball which makes (1.4) tractable.

In greater detail, oBB is a Lipschitz derivative based approach that uses an overlapping covering of balls rather than rectangular partitions. The main idea behind oBB is to recursively split an initial ball covering the domain \mathcal{D} into sub-balls until we find a ball (or balls) of sufficiently small size containing the global minimiser of f over \mathcal{D} . Using (1.4), oBB is able to obtain lower bounds on the minimum of f over each ball which can then be used to discard balls that cannot contain the global minimiser, i.e. balls whose lower bound is greater than the smallest upper bound. Each ball of radius r is split into 3^n overlapping sub-balls of half-radius $r/2$ centred at the vertices of a hypercubic tessellation of edge-length r/\sqrt{n} around the centre of the ball. This ensures that the sub-balls entirely cover the original ball with a constant amount of overlap irrespective of the original ball's radius. A detailed description of oBB is given as Sect. 3.1 when the latter is run on one (master) processor core.

As one would expect, testing the proposed bounds in oBB we find that in general, due to additional problem information being employed in the model, second order models yield better lower bounds on the objective function compared to first order models and hence can potentially lead to better branch and bound algorithms for Lipschitz optimization.

In general, Lipschitz-based lower bounding subproblems are non-convex, and branch and bound algorithms require their global solution (an NP-hard problem over boxes, see Theorem 2 in [23]). This is usually achieved using techniques such as vertex enumeration, interval arithmetic, convexification or problem specific constructs [29] but these may not be flexible enough or suitably scalable for the purpose of generic problem solvers. The approaches in Evtushenko [10] and Evtushenko and Posypkin [11, 12] and in oBB allow global solution for both first and second order models by minimizing these models over Euclidean balls rather than boxes, which ensures that the subproblems can be solved in polynomial time. Evtushenko [10], Evtushenko and Posypkin [11, 12] propose the use of a non-uniform mesh and employ the global solutions over balls to exclude elements in such a non-uniform rectangular partition. While their algorithm is already in a parallel framework it only uses first order models. Gaviano and Lera [17] devise a similar algorithm which also excludes elements from a non-uniform rectangular partition using zeroth order models. oBB uses second order models on overlapping balls leading to an overlapping covering of the domain (as opposed to the rectangular partition used in [10–12]). Thus the oBB approach leads to more computational effort as well as potential doubling of work so parallelisation is both crucial and challenging for obtaining good performance.

Similarly to other branch and bound algorithms, there is also the curse of dimensionality which is made worse by the high number of balls in each oBB covering. At each iteration, oBB splits a ball into 3^n smaller sub-balls (with constant overlap) whereas traditional branch and bound splits a box into only two larger sub-boxes. In the worst case both algorithms can be said to perform comparably, with each ball in oBB being split into 3^n sub-balls, compared to each box being split into 2^n sub-boxes for traditional branch and bound. However, as both algorithms use disparate coverings with subdomains of different sizes, it is difficult to compare them directly in general.

Due to the curse of dimensionality, many parallel branch and bound algorithms over boxes have been proposed in the literature (see [1, 2, 6, 8, 13, 31] and the survey by Gendron and Crainic [18]). Gendron and Crainic [18], Crainic et al. [8] have classified the main approaches into two classes: Type I and Type II parallelism that correspond to forms of data parallelism and task parallelism respectively. In Type I parallelism operations on subproblems (e.g. bounding) are conducted in parallel whereas the branch and bound tree is explored in serial (i.e. by one processor). In Type II parallelism by contrast, the tree itself is explored in parallel by many processors. It should be noted that while branch and bound algorithms are conceptually thought of as exploring a tree, for reasons of efficiency they are often

implemented numerically as a priority queue [8]. In order to parallelise oBB, we develop parallel algorithms using both data parallel (Type I) and task parallel (Type II) paradigms. Our main contribution here is to develop an effective task parallel variant of oBB using novel hashing techniques that enable efficient communication, essentially removing the doubling of work entirely. Additionally, we address the problem of balancing the load between processors by implementing an effective load balancing strategy.

The layout of the paper is as follows. First order lower bound estimates are given in Sect. 2.1 and second order lower bound estimates in Sect. 2.2, with numerical results presented in Sect. 2.3. We then consider the two main paradigms for parallelising the oBB algorithm, data parallel (bounds in parallel) in Sect. 3.1 and task parallel (tree in parallel) in Sect. 3.2 with numerical results in Sect. 3.3. Finally, we draw conclusions in Sect. 4.

2 Improving Lipschitz lower bounds

Let us first consider devising more accurate lower bound estimates for Lipschitz based branch and bound algorithms. We will therefore begin this section by looking at improved estimates for the first order lower bound (1.2) and then extend some of these ideas to the second order lower bound (1.3). It should be noted that the first order lower bound (1.2) and refinements are well known in the Lipschitz derivative optimization literature, see for example, [3,4,10–12,16,24,25,27,35] but the use of the cubic lower bound (1.3) within a global optimization context is recent (to the best of our knowledge).

2.1 First order lower bounds

As far as we are aware, there are two principal approaches in the literature which provide suitable estimates for the first order lower bound (1.2) and we will briefly describe these before discussing alternative approaches. The approach taken to estimate the gradient Lipschitz constant in [16] was to bound the norm of the Hessian over a suitable domain using interval arithmetic. Evtushenko and Posypkin [11, 12], amongst others, replace the negative Lipschitz constant by a lower bound on the spectrum of the Hessian, $\lambda_{\min}(H(x))$, for x in some interval, which they claim yields a more accurate estimate. They approximate $\lambda_{\min}(H(x))$ using Gershgorin's Theorem, but other approximations to $\lambda_{\min}(H(x))$, for x in some domain, have been proposed in the literature. Floudas ([14], Section 12.4) provides a useful summary of such approximations to convexify the objective function in the context of his branch and bound algorithm. In this section, we show that some of the estimates from [14] are more accurate than the Lipschitz constant estimates considered in [16] and estimates using Gershgorin's Theorem in [12].

We assume the following about problem (1.1) throughout this section:

AF 1 *The objective function $f: \mathcal{C} \rightarrow \mathbb{R}$ is twice continuously differentiable, where $\mathcal{C} \subset \mathbb{R}^n$ is a sufficiently large open set containing the convex, compact domain \mathcal{D} .*¹

Let us start by showing why lower bounds on $\lambda_{\min}(H(x))$ can be used in place of the gradient Lipschitz constant $-L_g$ in (1.2). To this end, define for some compact domain \mathcal{B}

$$\lambda_{\min}^{\mathcal{B}}(H) := \min_{\xi \in \mathcal{B}} \lambda_{\min}(H(\xi)). \quad (2.1)$$

¹ Note that we need a larger set here as the balls in our overlapping covering extend outside the domain during the initial subdivisions.

Lemma 2.1 [12] *Let AF1 hold. Suppose $\mathcal{B} \subset \mathcal{C}$ is a convex, compact subdomain and $x_{\mathcal{B}} \in \mathcal{B}$.² Then, for any $x \in \mathcal{B}$ we have*

$$f(x) \geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{\lambda_{\min}^{\mathcal{B}}(H)}{2} \|x - x_{\mathcal{B}}\|_2^2. \tag{2.2}$$

Proof For all $x, x_{\mathcal{B}} \in \mathcal{B}$ and some $\xi(x) \in \mathcal{B}$ the first order Taylor expansion with the Lagrange form for the remainder gives

$$\begin{aligned} f(x) &= f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(\xi)(x - x_{\mathcal{B}}) \\ &= f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2} \frac{(x - x_{\mathcal{B}})^T H(\xi)(x - x_{\mathcal{B}})}{(x - x_{\mathcal{B}})^T (x - x_{\mathcal{B}})} (x - x_{\mathcal{B}})^T (x - x_{\mathcal{B}}) \\ &\geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{\lambda_{\min}(H(\xi))}{2} \|x - x_{\mathcal{B}}\|_2^2 \\ &\geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{\lambda_{\min}^{\mathcal{B}}(H)}{2} \|x - x_{\mathcal{B}}\|_2^2 \end{aligned}$$

where the last two inequalities follow from the fact that the Rayleigh quotient reaches its minimum at the smallest eigenvalue and from (2.1), respectively. \square

We can therefore use any lower bound on $\lambda_{\min}^{\mathcal{B}}(H)$ in place of $-L_g(\mathcal{B})$ in (1.2). In particular, we consider the following possible lower bounds on $\lambda_{\min}^{\mathcal{B}}(H)$ from ([14], Section 12.4) which all require the following bounds on the Hessian.

Definition 2.1 Let AF1 hold. Let $h_{ij}(\xi)$ denote the elements of the Hessian matrix $H(\xi)$ of f . Furthermore, let $\underline{H} = (\underline{h}_{ij})_{1 \leq i, j \leq n}$, $\overline{H} = (\overline{h}_{ij})_{1 \leq i, j \leq n}$ be such that for all $i, j = 1, \dots, n$

$$h_{ij} \leq h_{ij}(\xi) \leq \overline{h}_{ij} \tag{2.3}$$

for all ξ in a convex, compact subdomain \mathcal{B} .

Such elementwise lower and upper bounds (2.3) can be obtained, for example, using interval arithmetic.

Theorem 2.1 [14] *Let AF1 hold. Given the elementwise bounds $\underline{h}_{ij}, \overline{h}_{ij}$ and corresponding matrices $\underline{H}, \overline{H}$ in (2.3), the following lower bounds for $\lambda_{\min}^{\mathcal{B}}(H)$ in the bound (2.2) hold:*

i) Gershgorin’s Theorem (**Ger**):

$$\lambda_{\min}^{\mathcal{B}}(H) \geq \min_i \left[\underline{h}_{ii} - \sum_{j \neq i} \max \{ |\underline{h}_{ij}|, |\overline{h}_{ij}| \} \right] \tag{2.4}$$

ii) E-Matrix Diagonal (**Ediag**):

$$\lambda_{\min}^{\mathcal{B}}(H) \geq \lambda_{\min}(H_M) - \rho(\Delta H) \tag{2.5}$$

where $\lambda_{\min}(H_M)$ denotes the smallest eigenvalue of the midpoint matrix $H_M := \frac{\overline{H} + \underline{H}}{2}$ and $\rho(\Delta H)$ the spectral radius of the radius matrix $\Delta H := \frac{\overline{H} - \underline{H}}{2}$.

² Note that \mathcal{B} does not need to be convex provided all line segments from $x_{\mathcal{B}}$ to x are contained in \mathcal{B} , i.e. if \mathcal{B} is star-convex with star-centre $x_{\mathcal{B}}$.

iii) *E*-Matrix Zero (**E0**):

$$\lambda_{\min}^{\mathcal{B}}(H) \geq \lambda_{\min}(\widehat{H_M}) - \rho(\widehat{\Delta H}) \tag{2.6}$$

where the modified radius matrix $\widehat{\Delta H}$ is ΔH with zeros on the diagonal and the modified midpoint matrix $\widehat{H_M}$ is H_M with h_{ii} on the diagonal.

iv) Lower Bounding Hessian (**lbH**):

$$\lambda_{\min}^{\mathcal{B}}(H) \geq \lambda_{\min}(L) \tag{2.7}$$

where the lower bounding Hessian $L = (l_{ij})$ is defined as

$$l_{ij} = \begin{cases} h_{ii} + \sum_{k \neq i} \frac{h_{ik} - \bar{h}_{ik}}{2} & \text{if } i = j \\ \frac{h_{ij} + \bar{h}_{ij}}{2} & \text{if } i \neq j \end{cases}$$

v) Hertz's Method (**HZ**):

$$\lambda_{\min}^{\mathcal{B}}(H) = \min_k \{\lambda_{\min}(H_k)\} \tag{2.8}$$

where the vertex matrices H_k are defined as follows: Let $x \in \mathbb{R}^n$, then there are 2^{n-1} possible combinations for the signs of the $x_i x_j$ products ($i \neq j$). For the k th such combination, define the vertex matrix $H_k = (h_{ij}^k)$ where

$$h_{ij}^k = \begin{cases} h_{ii} & \text{if } i = j, \\ h_{ij} & \text{if } x_i x_j \geq 0, i \neq j \\ \bar{h}_{ij} & \text{if } x_i x_j < 0, i \neq j \end{cases}$$

Proof See ([14], Section 12.4) for proofs of the above lower bounds (2.4)–(2.8). □

We also consider a lower bound on the best $-L_g(\mathcal{B})$ in (1.2), given in the following Theorem.

Theorem 2.2 (Norm of the Hessian (**Norm**)) *Let 1 hold. Suppose $\mathcal{B} \subset \mathcal{C}$ is a convex, compact subdomain and $x_{\mathcal{B}} \in \mathcal{B}$. Then, for any $x \in \mathcal{B}$, the first order lower bound (1.2) holds.³ Furthermore, a lower bound for the best $-L_g(\mathcal{B})$ in (1.2) is given by*

$$-L_g(\mathcal{B}) \geq -\sqrt{\sum_{ij} \max\{|h_{ij}|, |\bar{h}_{ij}|\}}^2} \tag{2.9}$$

where the elementwise bounds $\underline{h}_{ij}, \bar{h}_{ij}$ are defined in (2.3).

Proof (1.2) is a well-known consequence of first order Taylor expansions; see for example Theorem 3.1.4 in [7]. Note that $\|M\|_2 \leq \|M\|_F$ for any matrix M . We have from Taylor's theorem to first order and Cauchy-Schwarz that for any $x, y \in \mathcal{B}$

$$\begin{aligned} \|g(x) - g(y)\|_2 &\leq \left\| \int_0^1 H(y + \tau(x - y))(x - y) d\tau \right\|_2 \\ &\leq \max_{0 \leq \tau \leq 1} \|H(y + \tau(x - y))\|_2 \|x - y\|_2 \end{aligned}$$

³ Note that if \mathcal{B} is not assumed to be compact, then (1.2) still holds provided the gradient g is Lipschitz continuous on the convex subdomain \mathcal{B} and $f \in C^1(\mathcal{B})$.

$$\begin{aligned}
 &\leq \max_{0 \leq \tau \leq 1} \|H(y + \tau(x - y))\|_F \|x - y\|_2 \\
 &= \max_{0 \leq \tau \leq 1} \left(\sum_{ij} [H(y + \tau(x - y))]_{ij}^2 \right)^{1/2} \|x - y\|_2 \\
 &\leq \left(\sum_{ij} \max \{ |h_{ij}|^2, |\bar{h}_{ij}|^2 \} \right)^{1/2} \|x - y\|_2 \\
 &= \left(\sum_{ij} \max \{ |h_{ij}|, |\bar{h}_{ij}| \}^2 \right)^{1/2} \|x - y\|_2.
 \end{aligned}$$

Thus the gradient g is Lipschitz continuous on a compact domain \mathcal{B} with ℓ^2 -norm Lipschitz constant $\sqrt{\sum_{ij} \max\{|h_{ij}|, |\bar{h}_{ij}|\}^2}$. In particular, this means that for the best gradient Lipschitz constant $L_g(\mathcal{B})$, we have for all $x \in \mathcal{B}$

$$L_g(\mathcal{B}) \leq \sqrt{\sum_{ij} \max \{ |h_{ij}|, |\bar{h}_{ij}| \}^2}.$$

□

If we look at the computational cost of the estimation approaches given in (2.4)–(2.8) and (2.9) (and exclude the cost of calculating the Hessian bounds h_{ij}, \bar{h}_{ij}) we can show that **Ger** is an $O(n^2)$ method (i.e. it requires $O(n^2)$ floating point operations). **Eidag**, **E0**, **IbH** require the calculation of one or two extreme eigenvalues and **Hz** requires 2^{n-1} leftmost eigenvalues. Assuming standard methods for calculating all the eigenvalues of a matrix, **Eidag**, **E0**, **IbH** would all be $O(n^3)$ methods and **Hz** would be an $O(2^{n-1}n^3)$ method. In practice, extreme eigenvalues of dense matrices are usually obtained in $O(n^{2+v})$ flops, where $v < 1$, e.g. using the power method. Calculating **Norm** requires squaring n^2 entries and so is an $O(n^2)$ method.

2.2 Second order lower bounds

In Sect. 2.1, we considered replacing the gradient Lipschitz constant in the first order lower bound (1.2) by an estimate of the smallest eigenvalue of the Hessian. In this section we will show that, to an extent, a similar approach is also possible for the second order lower bound (1.3) and we can replace the Hessian Lipschitz constant by an estimate of the smallest eigenvalue of the derivative tensor. Before we describe this in detail we need to introduce some tensor eigenvalue notation.

Let $T \in \mathbb{R}^{n \times n \times n}$ denote a third order tensor, which being a generalisation of a matrix to three indices, is a 3-dimensional array. As with matrices, t_{ijk} denotes the (i, j, k) th component (i.e. element in the array) of the tensor T . Furthermore, a tensor T is called symmetric (sometimes supersymmetric) if $t_{\sigma(i)\sigma(j)\sigma(k)} = t_{ijk}$ for any permutation σ of the indices (i, j, k) . This is the natural generalisation of a symmetric matrix to tensors. For a vector $x \in \mathbb{R}^n$, the multiplication of a tensor T three times on the right by x is denoted by

$$Tx^3 := \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n t_{ijk} x_i x_j x_k.$$

Let $\|T\|_F$ denote the Frobenius norm for the tensor T defined as

$$\|T\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n t_{ijk}^2.$$

We have from [28] that the *multilinear Rayleigh quotient* for the ℓ^3 -norm is given by

$$\frac{Tx^3}{\|x\|_3^3}$$

where $\|\cdot\|_3$ is the ℓ^3 -norm for vectors. The ℓ^3 -eigenvalues (or H-eigenvalues) of T are then defined as the stationary points of the multilinear Rayleigh quotient. In particular, this means that the smallest ℓ^3 -eigenvalue of T , $\lambda_{\min}^{\ell^3}(T)$ is given by⁴

$$\lambda_{\min}^{\ell^3}(T) = \min_{x \neq 0} \frac{Tx^3}{\|x\|_3^3}. \tag{2.10}$$

We assume the following about problem (1.1) throughout this section:

AF 2 *The objective function $f : \mathcal{C} \rightarrow \mathbb{R}$ is thrice continuously differentiable, where $\mathcal{C} \subset \mathbb{R}^n$ is a sufficiently large open set containing the convex, compact domain \mathcal{D} .*

We are now in a position to show why lower bounds on the spectrum of the derivative tensor can be used in place of the Hessian Lipschitz constant L_H in (1.3). To this end, let

$$T(x) := \nabla_{xxx} f(x)$$

denote the third order derivative tensor of $f(x)$ and note that it is symmetric by construction. Define for some compact domain \mathcal{B}

$$\lambda_{\min}^{\ell^3, \mathcal{B}}(T) := \min_{\xi \in \mathcal{B}} \lambda_{\min}^{\ell^3}(T(\xi)). \tag{2.11}$$

Lemma 2.2 *Let AF 2 hold. Suppose $\mathcal{B} \subset \mathcal{C}$ is a convex, compact subdomain and $x_{\mathcal{B}} \in \mathcal{B}$. Then, for any $x \in \mathcal{B}$ we have*

$$f(x) \geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) + \begin{cases} \frac{\lambda_{\min}^{\ell^3, \mathcal{B}}(T)}{6} \|x - x_{\mathcal{B}}\|_2^3 & \text{if } \lambda_{\min}^{\ell^3, \mathcal{B}}(T) \leq 0, \\ \frac{\lambda_{\min}^{\ell^3, \mathcal{B}}(T)}{6} n^{-1/2} \|x - x_{\mathcal{B}}\|_2^3 & \text{if } \lambda_{\min}^{\ell^3, \mathcal{B}}(T) > 0. \end{cases} \tag{2.12}$$

Proof First of all, in order to use ℓ^3 -eigenvalues in (2.12) we require relations between the ℓ^2 and ℓ^3 vector norms. It is a standard result that for any $p > r > 0$

$$\|x\|_p \leq \|x\|_r \leq n^{(1/r-1/p)} \|x\|_p$$

for any $x \in \mathbb{R}^n$ and in particular this means that

$$\|x\|_3 \geq n^{-1/6} \|x\|_2,$$

⁴ Note that one can instead use the alternative definition of ℓ^2 -eigenvalues that are the stationary points of the multilinear Rayleigh quotient for the ℓ^2 -norm, $Tx^3/\|x\|_2^3$ [28] and then the smallest ℓ^2 -eigenvalue of T , $\lambda_{\min}^{\ell^2}(T)$ would be given by $\lambda_{\min}^{\ell^2}(T) = \min_{x \neq 0} Tx^3/\|x\|_2^3$. However, we will not use ℓ^2 -eigenvalues here for reasons that will become clear later.

$$\|x\|_3 \leq \|x\|_2 \tag{2.13}$$

for any $x \in \mathbb{R}^n$.

Now, for $x = x_{\mathcal{B}}$ the claim in the theorem is trivial, so w.l.o.g. assume $x \neq x_{\mathcal{B}}$. Then for all $x, x_{\mathcal{B}} \in \mathcal{B}$ and some $\xi(x) \in \mathcal{B}$, the second order Taylor expansion with the Lagrange form for the remainder gives

$$\begin{aligned} f(x) &= f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) + \frac{1}{6}T(\xi)(x - x_{\mathcal{B}})^3 \\ &= f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) + \frac{1}{6} \frac{T(\xi)(x - x_{\mathcal{B}})^3}{\|x - x_{\mathcal{B}}\|_3^3} \|x - x_{\mathcal{B}}\|_3^3 \\ &\geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) + \frac{\lambda_{\min}^{\ell^3}(T(\xi))}{6} \|x - x_{\mathcal{B}}\|_3^3 \\ &\geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) + \frac{\lambda_{\min}^{\ell^3, \mathcal{B}}(T)}{6} \|x - x_{\mathcal{B}}\|_3^3 \\ &\geq f(x_{\mathcal{B}}) + (x - x_{\mathcal{B}})^T g(x_{\mathcal{B}}) + \frac{1}{2}(x - x_{\mathcal{B}})^T H(x_{\mathcal{B}})(x - x_{\mathcal{B}}) \\ &\quad + \begin{cases} \frac{\lambda_{\min}^{\ell^3, \mathcal{B}}(T)}{6} \|x - x_{\mathcal{B}}\|_2^3 & \text{if } \lambda_{\min}^{\ell^3, \mathcal{B}}(T) \leq 0, \\ \frac{\lambda_{\min}^{\ell^3, \mathcal{B}}(T)}{6} n^{-1/2} \|x - x_{\mathcal{B}}\|_2^3 & \text{if } \lambda_{\min}^{\ell^3, \mathcal{B}}(T) > 0 \end{cases} \end{aligned}$$

using (2.10), (2.11) and (2.13) respectively. □

We can therefore use any (suitably scaled) lower bound on $\lambda_{\min}^{\ell^3, \mathcal{B}}(T)$ in place of $-L_H(\mathcal{B})$ in (1.3).⁵ In Sect. 2.1, Theorem 2.1 (2.4)–(2.8) and Theorem 2.2 (2.9) give several different approaches to obtain lower bounds on the smallest eigenvalue in the case of a Hessian matrix. We will now show which of these estimation approaches generalises to the case of a third order derivative tensor. While there are ℓ^3 -eigenvalue algorithms that are guaranteed to converge to the smallest eigenvalue, these are only applicable to tensors with non-negative (or equivalently non-positive) entries [22]. Unfortunately, the tensor generalisations of the matrices required for the lower bounding strategies presented in (2.5)–(2.8), namely the E -matrix, Lower bounding Hessian and Hertz method have both positive and negative entries in general. However, the generalisation of Gershgorin’s Theorem [34] does not require an eigenvalue algorithm and we can therefore generalise Theorem 2.1 (2.4) to tensors. We first need the following definition before we can give the generalised theorem.

Definition 2.2 Let AF 2 hold. Let $t_{ijk}(\xi)$ denote the elements of the third order derivative tensor $T(\xi)$. Furthermore, let $\underline{T} = (\underline{t}_{ijk})_{1 \leq i, j, k \leq n}$, $\bar{T} = (\bar{t}_{ijk})_{1 \leq i, j, k \leq n}$ be such that for all $i, j, k = 1, \dots, n$

$$\underline{t}_{ijk} \leq t_{ijk}(\xi) \leq \bar{t}_{ijk} \tag{2.14}$$

for all ξ in a convex, compact subdomain \mathcal{B} .

Once again, the elementwise lower and upper bounds (2.14) can be obtained using interval arithmetic.

⁵ Note that an analogous result holds for ℓ^2 -eigenvalues. Unfortunately, to the best of our knowledge, there are no known eigenvalue algorithms that are guaranteed to converge to the smallest ℓ^2 -eigenvalue but it is possible to use generalisations of the power method using multiple starting points [22, 41]. However, this is not reliable as (1.3) requires a bound on the smallest eigenvalue and using multiple starting points does not guarantee this. Furthermore, there is no generalisation of Gershgorin’s Theorem for ℓ^2 -eigenvalues, which is what we propose next for ℓ^3 -eigenvalues.

Theorem 2.3 (Gershgorin’s Theorem for the derivative Tensor (**Ger T**)) *Let AF 2 hold. Assuming the elementwise bounds $\underline{t}_{ijk}, \bar{t}_{ijk}$ in (2.14), $\lambda_{\min}^{\ell^3, \mathcal{B}}(T)$ in (2.12) can be bounded below by*

$$\lambda_{\min}^{\ell^3, \mathcal{B}}(T) \geq \min_i \left[\underline{t}_{iii} - \sum_{k \neq j \neq i} \max \{ |\underline{t}_{ijk}|, |\bar{t}_{ijk}| \} \right]. \tag{2.15}$$

Proof Let $\xi \in \mathcal{B}$ be arbitrary. We have from [34] that Gershgorin’s Theorem for tensors applied to the third order derivative tensor $T(\xi)$ gives

$$\begin{aligned} \lambda_{\min}^{\ell^3}(T(\xi)) &= \min_i \left[\underline{t}_{iii} - \sum_{k \neq j \neq i} |t_{ijk}(\xi)| \right] \\ &\geq \min_i \left[\underline{t}_{iii} - \sum_{k \neq j \neq i} \max \{ |\underline{t}_{ijk}|, |\bar{t}_{ijk}| \} \right] \end{aligned}$$

for any $\xi \in \mathcal{B}$. As $\lambda_{\min}^{\ell^3, \mathcal{B}}(T) = \min_{\xi \in \mathcal{B}} \lambda_{\min}^{\ell^3}(T(\xi))$ from (2.11), the result follows. \square

Additionally, we also have a bound on the Hessian Lipschitz constant in (1.3), an extension of the **Norm** bound (2.9) from Theorem 2.2.⁶

Theorem 2.4 (Norm of the derivative tensor (**Norm T**)) *Let AF 2 hold. Suppose $\mathcal{B} \subset \mathcal{C}$ is a convex, compact subdomain and $x_{\mathcal{B}} \in \mathcal{B}$. Then, for any $x \in \mathcal{B}$, the second order lower bound (1.3) holds.⁷ Furthermore, a lower bound for the best $-L_H(\mathcal{B})$ in (1.3) is given by*

$$-L_H(\mathcal{B}) \geq - \sqrt{\sum_{ijk} \max \{ |\underline{t}_{ijk}|, |\bar{t}_{ijk}| \}^2} \tag{2.16}$$

where the elementwise bounds $\underline{t}_{ijk}, \bar{t}_{ijk}$ are defined as in (2.14).

Proof (1.3) is a well-known consequence of second order Taylor expansions; see for example Theorem 3.1.5 in [7]. Note that as in the matrix case, $\|T\|_2 \leq \|T\|_F$ for any tensor T (see Lemma 6.1 in [16], for a proof). We have from Taylor’s theorem to first order and Cauchy-Schwarz that for any $x, y \in \mathcal{B}$

$$\begin{aligned} \|H(x) - H(y)\|_2 &\leq \left\| \int_0^1 T(y + \tau(x - y))(x - y) d\tau \right\|_2 \\ &\leq \max_{0 \leq \tau \leq 1} \|T(y + \tau(x - y))\|_F \|x - y\|_2 \\ &= \max_{0 \leq \tau \leq 1} \left(\sum_{ijk} [T(y + \tau(x - y))]_{ijk}^2 \right)^{1/2} \|x - y\|_2 \\ &\leq \left(\sum_{ijk} \max \{ |\underline{t}_{ijk}|, |\bar{t}_{ijk}| \}^2 \right)^{1/2} \|x - y\|_2. \end{aligned}$$

⁶ Note that this bound appears in Section 6.2 of Fowkes et al. [16] but it is incorrect there.

⁷ Note that if \mathcal{B} is not assumed to be compact, then (1.3) still holds provided the Hessian H is Lipschitz continuous on the convex subdomain \mathcal{B} and $f \in C^2(\mathcal{B})$.

Thus the Hessian H is Lipschitz continuous on a compact domain \mathcal{B} with ℓ^2 -norm Lipschitz constant $\sqrt{\sum_{ijk} \max\{|L_{ijk}|, |\bar{T}_{ijk}|\}^2}$. In particular, this means that for the best Hessian Lipschitz constant $L_H(\mathcal{B})$, we have

$$L_H(\mathcal{B}) \leq \sqrt{\sum_{ijk} \max\{|L_{ijk}|, |\bar{T}_{ijk}|\}^2}.$$

□

Looking at the computational cost of the second order estimation approaches **Ger T** and **Norm T** given in (2.15), (2.16) (and excluding the cost of calculating the tensor bounds L_{ijk}, \bar{T}_{ijk}) we can see that they are $O(n^3)$ methods since each requires summing or squaring n^3 elements.

2.3 Numerical results

The overlapping branch and bound algorithm (oBB), namely Algorithm 2.1 from Fowkes et al. [16], is especially suited to testing the first and second order estimation approaches from Sects. 2.1 and 2.2. As oBB is exactly Algorithm 3.1 from Sect. 3.1 in which all the worker calculations are performed by the master, we will only briefly outline it here and postpone a detailed description to Sect. 3.1. The algorithm is structured in much the same way as most standard branch and bound algorithms: It starts with a ball covering the domain and recursively subdivides it into overlapping balls, bounding each ball and discarding balls that cannot contain a global minimiser until the global minimum is located. The branching subdivides each ball into 3^n half-sized overlapping sub-balls that cover the original ball and have a fixed amount of overlap. The bounding uses the first and second order Lipschitz-based lower bounds (1.2), (1.3) but can also accommodate the eigenvalue-based lower bounds (2.2), (2.12) all of which it can solve in polynomial due to its use of overlapping balls.

We test the first and second order estimation approaches on test sets of

- 1) Random polynomials
- 2) Random radial basis functions (RBFs)

which we will describe in turn. The aim of the numerical experiments is to test which estimation approach gives the best oBB performance in terms of runtime. This gives an indirect indication of the accuracy of the estimation approach.

Random Polynomials [12] This is a collection of bound constrained global optimization problems with polynomial objective functions and randomly generated coefficients. The polynomial objective functions used are of the form

$$f(x) = \sum_{i=1}^n 10x_i^m + \sum_{p \in \mathcal{P}} a_p x_{i_1}^{p_1} \dots x_{i_n}^{p_n} \tag{2.17}$$

where m is an even polynomial degree and $\mathcal{P} = \{(p_1, \dots, p_n) : p_i \in \mathbb{Z}_+, \sum_{i=1}^n p_i \leq m-1\}$ is the set of n -tuples corresponding to the powers of the monomials. The randomly generated coefficients a_p are uniformly distributed in $[0, 10]$. Let $|\mathcal{P}| = \binom{m-1+n}{n}$ be the number of n -tuples in \mathcal{P} . Evtushenko and Posypkin [12] then observe that for an even m , the global optimiser lies in the box $[-|\mathcal{P}|, |\mathcal{P}|]^n$ and this is therefore taken to be the search domain. Following [12], we set the following following values for m and n in (2.17):

- Series 1 : $n = 3, m = 4$
 - Series 2 : $n = 3, m = 6$
 - Series 3 : $n = 4, m = 4$
- (2.18)

and test 10 realisations of (2.17) for each series. The bounds required by our estimation approaches, namely $\underline{h}_{ij}, \bar{h}_{ij}$ in (2.3) on the Hessians and $\underline{t}_{ijk}, \bar{t}_{ijk}$ in (2.14) on the derivative tensors of the polynomials are calculated using our own implementation of standard interval arithmetic (see e.g. Section 11 of Neumaier [29]).

Random RBFs This is a RBF test set similar to the one above for random polynomials. We will use cubic spline RBF objective functions of the form

$$f(x) = \mu_0 + \sum_{i=1}^n \mu_i x_i + \sum_{j=1}^{N(m,n)} \lambda_j \|x - x^j\|_2^3 \tag{2.19}$$

where μ_i, λ_j are coefficients of the linear and radial terms respectively and $N(m, n) := \binom{m-1+n}{n}$ is a given number of centres $x^j \in \mathbb{R}^n$. As before, we let the coefficients μ_i, λ_j be random, that is uniformly distributed in the interval $[0, 10]$. We will use the same values for m and n in (2.19) as those for random polynomials given in (2.18) and test 10 realisations of (2.19) for each series. Note that we choose $N(m, n) = \binom{m-1+n}{n}$ so that we have the same number of terms in the RBFs as in the polynomials above (up to a constant). We also take the box $[-N(m, n), N(m, n)]^n$ as the search domain so that the search regions for the RBFs are the same as for the polynomials. The bounds $\underline{h}_{ij}, \bar{h}_{ij}$ in (2.3) on the Hessians and $\underline{t}_{ijk}, \bar{t}_{ijk}$ in (2.14) on the derivative tensors of the RBFs are calculated using a more accurate interval arithmetic type approach (see Section 6.2 of Fowkes et al. [16], for details).

As we are interested in the relative performance of the first and second order estimation approaches, we will look at runtime performance profiles for both the random polynomial and RBF test sets described above (for the definition of performance profiles, see [9]; see also [40], p. 203, for the similar notion of operating characteristics first proposed by Grishagin [19]). To this end, we ran a Python-based serial implementation of oBB to an absolute tolerance of 10^{-6} for the global minimum using one of the estimation approaches given in (2.4)–(2.8) and (2.9). If the algorithm did not complete a run in 24h then that run was considered a failure. The hardware used was part of the ECDF Eddie cluster using a single 2.4GHz Intel Xeon E5645 processor core with 2GB of RAM for each random polynomial or RBF realisation.

2.3.1 First order lower bounds

Let us first consider the numerical performance of the first order estimation approaches **Ger**, **Eidag**, **E0**, **lbH**, **HZ** and **Norm** given in (2.4)–(2.8) and (2.9). Figure 1 shows performance profiles of the total runtime for the first order estimation approaches on the random polynomials.

For clarity we consider two ranges of the performance ratio so we can clearly see the poorer estimates (**Norm** and **Eidag**) in the top of Fig. 1 and the better estimates (**HZ**, **lbH**, **E0** and **Ger**) in the bottom of Fig. 1, which is a close-up of the top figure. From the top of Fig. 1 we can see that **Norm** is by far the weakest approach, in fact the algorithm only finds the global minimum within 24h in a third of the problems tested. For all the other estimates, the global minimum is always found, although the **Eidag** approach also performs poorly. Looking at the better performing approaches in the bottom of Fig. 1, we can see that **HZ** is the best, presumably because it always calculates exactly the smallest eigenvalue of the

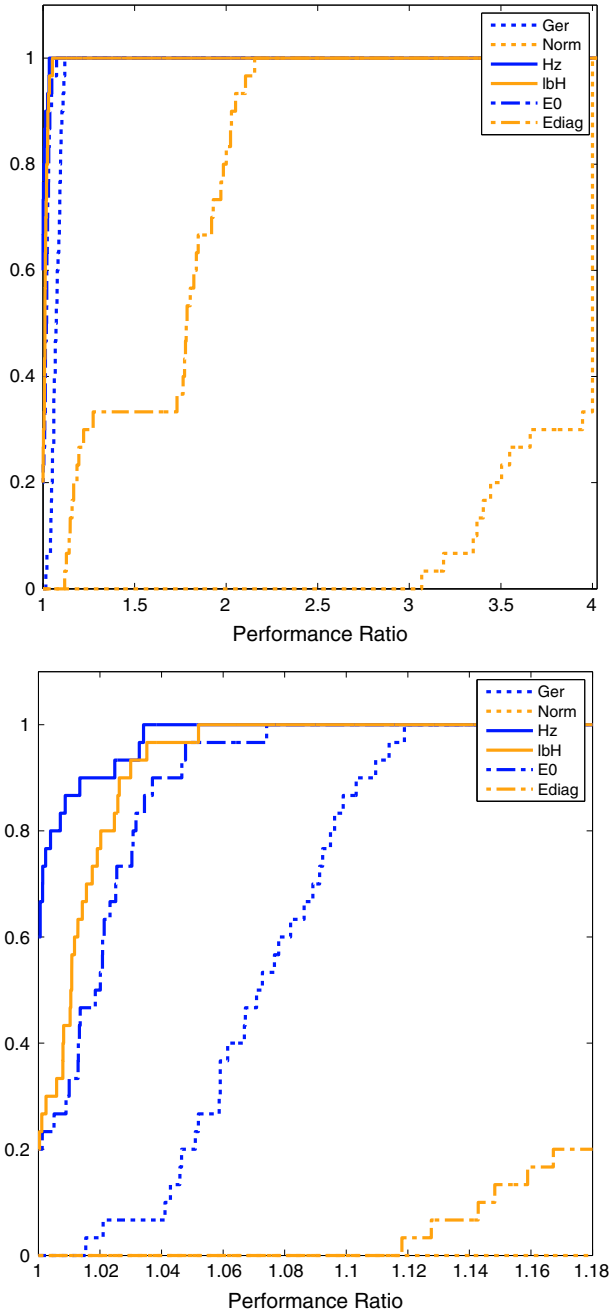


Fig. 1 Random polynomial runtime performance profiles (*top*) with a close up of the *top* figure (*bottom*) for the first order estimation approaches given in (2.4)–(2.8) and (2.9)

Hessian $H(\xi)$ for $\underline{H} \leq H(\xi) \leq \overline{H}$. However, this necessitates calculating the eigenvalues of 2^{n-1} matrices and while this is practical for two and three dimensional polynomials this will clearly be an issue in higher dimensions. With this in mind, the best approaches seem to

be **lbH** and **E0** which perform similarly well, followed by **Ger** which does not appear to be quite as good but nonetheless still shows reasonable performance.

Figure 2 shows performance profiles of the total runtime for the first order estimation approaches on the random RBFs. We can see from Fig. 2 that the performance profiles for random RBFs are very different from those for the random polynomials obtained in Fig. 1. In particular, **lbH** significantly outperforms all the other estimation approaches. **Ger**, **E0**, **H_z** and **Eidag** all perform similarly well and this is especially surprising as the **Eidag** approach showed poor performance on the random polynomials. The **Norm** approach is once again the weakest, although it performs somewhat better on the random RBFs.

It is evident from these numerical experiments that there is a need for several different estimation approaches as no single approach is superior. In particular, as the computational cost of the estimation approach is generally negligible compared to the cost of computing the bounds \underline{h}_{ij} , \bar{h}_{ij} , it is possible to have an adaptive algorithm that computes several estimates and uses the best one.

2.3.2 Second order lower bounds

We will now look at the numerical performance of the second order tensor Gershgorin and Norm approaches given in (2.15) and (2.16), respectively, on the random polynomial and RBF test sets. Figure 3 shows performance profiles of the total runtime for the second order estimation approaches on the random polynomials and RBFs. We can see in the top of Fig. 3 that the Gershgorin based estimate consistently outperforms the tensor norm approach for random polynomials. Although the algorithm always finds the global minimum using these estimates, the Gershgorin estimate yields faster and more accurate second order lower bounds. The situation, however, is completely reversed for random RBFs as we can see from the performance profiles in the bottom of Fig. 3. In this case the tensor norm based estimate outperforms the Gershgorin estimate and yields faster and more accurate bounds. Once again, this emphasises the need to compute several estimation approaches and use whichever is best.

2.3.3 Comparison of first versus second order bounds

Finally, to wrap up the discussion of finding better bounds, we compare both first and second order lower bounds in Fig. 4 by recalculating performance profiles of the total runtime for both. One can clearly see from the top of Fig. 4 that for random polynomials the second order lower bounds significantly outperform the first order ones, with the tensor Gershgorin approach clearly superior. This is perhaps not surprising as the second order lower bounds (2.12), (1.3) utilising second order derivative information are likely to be more accurate than the first order lower bounds (2.2), (1.2) which can only make use of first order information. However, the situation is not quite so simple for the random RBFs as we can see from the bottom of Fig. 4 where the first order lower bounding Hessian estimation approach actually outperforms the second order tensor Gershgorin approach. This is encouraging since it shows that in some cases first order bounds which are significantly cheaper to compute can be competitive with the more expensive second order bounds. The tensor norm approach is, however, evidently the best for random RBFs, outperforming all other approaches.

In conclusion, there is no single first or second order bound that is clearly superior across different objective functions. There are even instances where first order bounds outperform second order ones. The best strategy in our opinion is therefore to implement all the first and second order bounds within a branch and bound algorithm and adaptively choose which

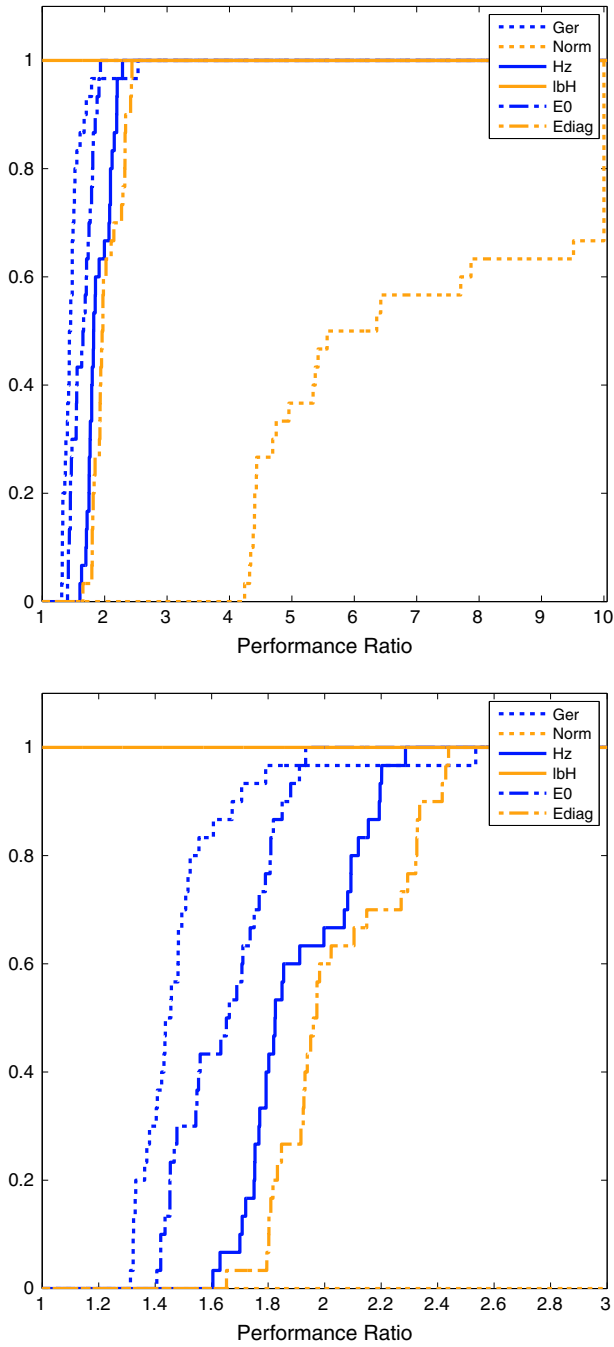


Fig. 2 Random RBF runtime performance profiles (*top*) with a close up of the *top* figure (*bottom*) for the first order estimation approaches given in (2.4)–(2.8) and (2.9)

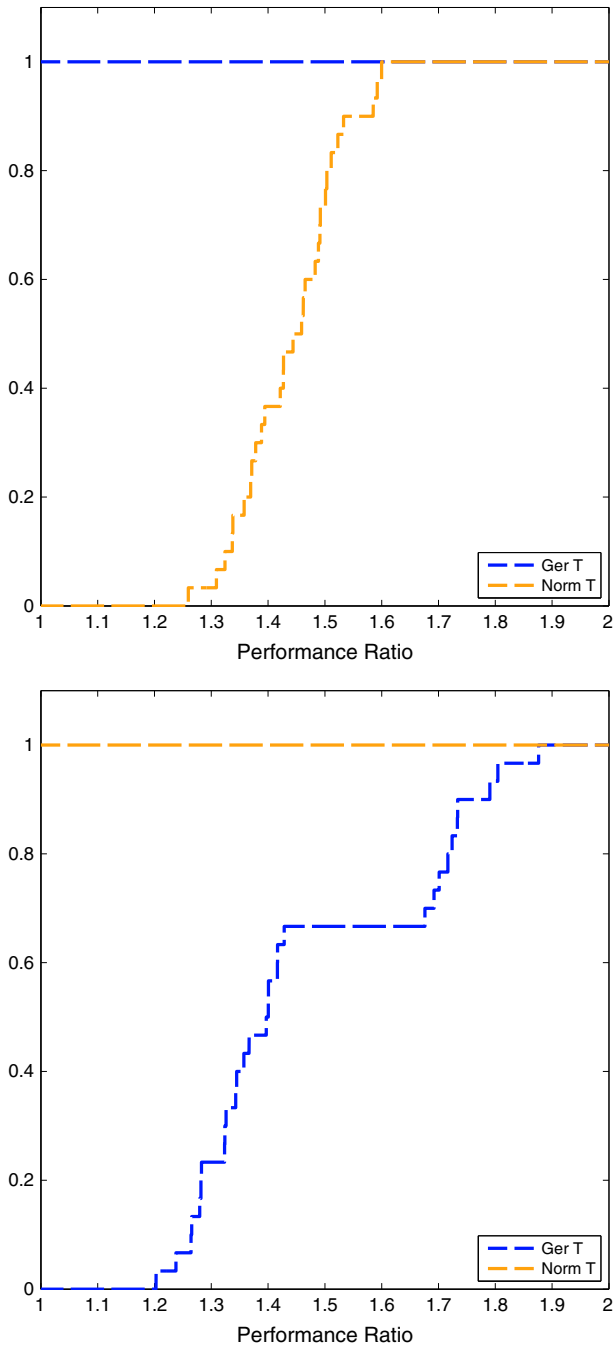


Fig. 3 Random Polynomial (*top*) and RBF (*bottom*) runtime performance profiles for the second order estimation approaches given in (2.15) and (2.16), respectively

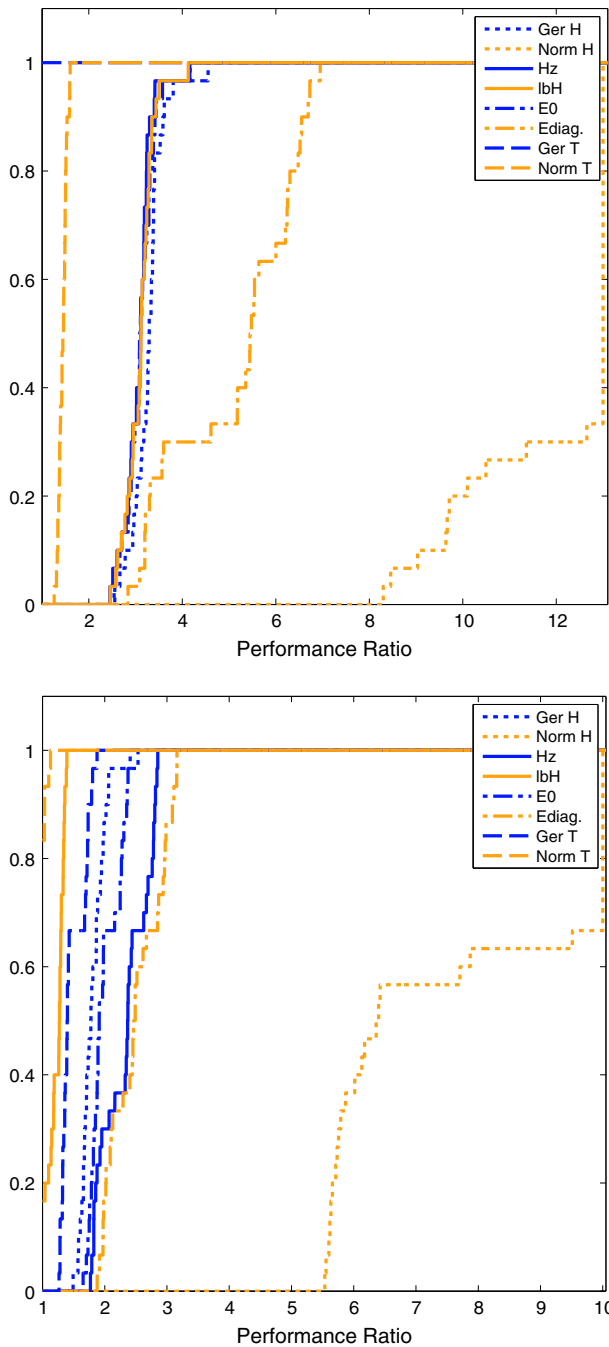


Fig. 4 Random Polynomial (*top*) and RBF (*bottom*) runtime performance profiles for the first order (*dotted, solid and dash-dotted lines*) and second order (*dashed lines*) estimation approaches

is best. For example, for the first few subproblems all possible bounds could be computed and the best two or three used throughout the rest of the computation. As the computational cost of calculating the lower bounds is negligible compared to the cost of calculating the bounds l_{ij}, \bar{h}_{ij} or $\underline{t}_{ijk}, \bar{t}_{ijk}$, such adaptive strategies are feasible and indeed recommended to maximise performance.

3 Parallelising overlapping branch and bound (oBB)

In Sect. 2 we considered improving the bounding in Lipschitz based branch and bound global optimization algorithms and tested our findings using an implementation of oBB, Algorithm 2.1 from [16]. In this section we will consider this implementation of oBB and show how it can be speeded up through parallelism. As mentioned in the introduction, there are two main approaches to parallelising branch and bound algorithms: data parallel, namely performing the bounding operations in parallel and task parallel, traversing the branch and bound tree in parallel [8, 18]. We consider applying these in turn to oBB in the following sections.

3.1 Data parallelism: bounds in parallel

The idea behind data parallelism of a branch and bound algorithm is to share the computational burden of calculating the bounds amongst many processor cores. Our implementation of this is a very straightforward master/worker approach. The master processor core runs the entire algorithm except for the calculations involved in obtaining bounds on each subdomain, which are (roughly) evenly divided amongst itself and the worker processors. It is immediately obvious that this type of parallelism will only be successful if there are many bounding calculations that can be performed independently at the same time and if these calculations are relatively expensive compared to the rest of the algorithm.

The oBB algorithm uses Euclidean balls as the subdomains since this allows the lower bounding subproblem (1.4) to be solved in polynomial time. However, this comes at a cost, as the rigorous variant of oBB requires that each ball is split into 3^n sub-balls which can very quickly become prohibitively large as the dimension n increases. Nevertheless, this lends itself well to data parallelism since at each step of the algorithm we have to bound around 3^n balls and these bounding operations can of course be done in parallel. This is the basis of the data parallel version of oBB, given below as Algorithm 3.1.

The algorithm solves (1.4) to obtain a lower bound $\underline{f}(\mathcal{B})$ on the objective function f over the subdomain \mathcal{B} , that is

$$\underline{f}(\mathcal{B}) := \min_{x \in \mathcal{B}} l_{\mathcal{B}}(x) \tag{3.1}$$

where $l_{\mathcal{B}}(x)$ can be any of the first or second order lower bounds given in (1.2), (2.2) and (1.3), (2.12), respectively. The upper bound $\bar{f}(\mathcal{B})$ on f over \mathcal{B} is simply the objective function f evaluated at a feasible point $x_F \in \mathcal{B}$, that is

$$\bar{f}(\mathcal{B}) := f(x_F). \tag{3.2}$$

It is important to note that if we run this algorithm on one master processor core, we recover the serial version of oBB.

Algorithm 3.1 Data Parallel Branch and Bound Algorithm

Master Processor

0. Initialisation:

- (a) Set $k = 0$ and t_{max} to be the maximum runtime.
- (b) Let \mathcal{B}_0 be a ball with centre $x_{\mathcal{B}} \in \mathcal{D}$ of sufficiently large radius to cover \mathcal{D} .
- (c) Let $\mathcal{L}_0 = \{\mathcal{B}_0\}$ be the initial set of balls.
- (d) Let $U_0 = \bar{f}(\mathcal{B}_0)$ be the initial upper bound for $\min_{x \in \mathcal{B}_0} f(x)$.
- (e) Let $L_0 = \underline{f}(\mathcal{B}_0)$ be the initial lower bound for $\min_{x \in \mathcal{B}_0} f(x)$.

1. While $U_k - L_k > \varepsilon$ and the *runtime* $< t_{max}$, repeat the following procedure:

- (a) Pruning: Remove from \mathcal{L}_k balls $\mathcal{B} \in \mathcal{L}_k$ such that $\underline{f}(\mathcal{B}) > U_k$.⁸
- (b) Branching: Choose $\mathcal{B} \in \mathcal{L}_k$ such that $\underline{f}(\mathcal{B}) = L_k$. Split \mathcal{B} into 3^n overlapping sub-balls $\mathcal{B}_1, \dots, \mathcal{B}_{3^n}$ according to the splitting rule in Sect. 2.2 of Fowkes et al. [16] and discard any sub-balls that lie entirely outside of \mathcal{D} . Let \mathcal{R}_k denote the set of remaining sub-balls and let $\mathcal{L}_{k+1} := (\mathcal{L}_k \setminus \{\mathcal{B}\}) \cup \mathcal{R}_k$.
- (c) Bounding: Partition \mathcal{R}_k into P subsets \mathcal{R}_k^p for $p \in \{1, \dots, P\}$ and distribute them amongst the P worker processors for bounding. Wait until all the bounds $\underline{f}(\mathcal{B}), \bar{f}(\mathcal{B})$ for $\mathcal{B} \in \mathcal{R}_k$ are received back.
- (d) Set $U_{k+1} := \min_{\mathcal{B} \in \mathcal{L}_{k+1}} \bar{f}(\mathcal{B})$.
- (e) Set $L_{k+1} := \min_{\mathcal{B} \in \mathcal{L}_{k+1}} \underline{f}(\mathcal{B})$.
- (f) Set $k = k + 1$.

2. Send termination signal to worker processors.

3. Return U_k as the estimate of the global minimum of $f(x)$ over \mathcal{D} .

Worker Processor p

1. Repeat the following procedure until termination signal is received:

- (a) Wait for a set of balls \mathcal{R}_k^p from the master processor.
- (b) When the set is received, calculate bounds $\underline{f}(\mathcal{B}), \bar{f}(\mathcal{B})$ for each ball $\mathcal{B} \in \mathcal{R}_k^p$ and send the bounds back to the master processor.

In step 1b, the algorithm splits each ball \mathcal{B} into 3^n overlapping sub-balls of half-radius $r(\mathcal{B})/2$ centred at the vertices of a hypercubic tessellation of edge length $r(\mathcal{B})/\sqrt{n}$ around the centre of the ball; see ([16], Sect. 2.2) for further details of this splitting rule.

3.2 Task parallelism: tree in parallel

In task parallelism of a branch and bound algorithm, the focus is on exploring the branch and bound tree in parallel. Conceptually, a branch and bound algorithm running on an arbitrary problem can be thought of as generating a tree. The nodes of the tree represent the subregions and the edges denote the regions they are split from. One can then think of having several processor cores generating different sections of the tree starting from different subregions. In the case of traditional branch and bound using boxes this is conceptually straightforward to implement as each subregion forms a distinct partition of the domain and any subregions

⁸ As correctly pointed out by an anonymous referee (and as used in Algorithm 3.2) one can optionally use $\underline{f}(\mathcal{B}) > U_k - \varepsilon$ as the condition for pruning which may allow the algorithm to discard more redundant balls.

split from it are also contained within that partition. All that is required is that the processor cores communicate the best upper bound found so far and balance the load, namely make sure the work is evenly distributed amongst the processor cores.

However, we are interested in parallelising oBB which uses overlapping balls rather than rectangular partitions. This makes the parallelisation more difficult since the balls do not form natural partitions. As such several processor cores can end up bounding and splitting the same promising ball, arrived at by repeatedly splitting different initial balls and so doubling of work can occur. Our solution to this problem is to essentially eliminate the doubling entirely through efficient communication using *hashing*, see Sect. 3.2.1.

We also need a strategy to balance the load between processor cores, i.e. the number of balls, or equivalently the number of subproblems, on each processor core. After due consideration we implemented a two tier strategy as most modern HPC clusters consists of a large number of nodes (i.e. sets of processors which share the same memory) interconnected by ethernet or infiniband switches. It therefore makes sense to load balance both within each node where communication via shared memory will be very efficient and across different nodes where communication via gigabit ethernet or infiniband will be relatively slow, see Sect. 3.2.2. The complete task parallel branch and bound algorithm is given below.

Algorithm 3.2 Task Parallel Branch and Bound Algorithm Master Processor

0. Initialisation

- (a) Set t_{max} to be the maximum runtime of the algorithm.
- (b) Let \mathcal{B} be a ball with centre $x_{\mathcal{B}} \in \mathcal{D}$ of sufficiently large radius to cover \mathcal{D} .
- (c) Split \mathcal{B} into 3^n overlapping sub-balls according to the splitting rule in Sect. 2.2 of Fowkes et al. [16] and discard any sub-balls that lie entirely outside of \mathcal{D} . Partition the remaining sub-balls into P subsets and distribute them amongst the p worker processors as sets \mathcal{L}^p for $p \in \{1, \dots, P\}$.
- (d) Let $\mathcal{R} = \emptyset$ be the initial ordered list of hashes of radii.
- (e) Let $\mathcal{C} = \emptyset$ be the initial ordered list of sets of hashes of centres with the same radius.

1. While $\mathcal{L}^p \neq \emptyset \forall p$ and the *runtime* $< t_{max}$, repeat the following procedure:

- (a) Asynchronously receive U_p and the size $|\mathcal{L}^p|$ of the set \mathcal{L}^p from all $p \in \{1, \dots, P\}$ worker processors.
- (b) Asynchronously send $U := \min_{p \in \{1, \dots, P\}} U_p$ to all P worker processors.
- (c) Hashing: Process lists of hashes received from worker processors, updating \mathcal{R} , the list of radius hashes,⁹ and \mathcal{C} , the list of ball-centre hashes, and inform the workers of any duplicate entries (see Sect. 3.2.1).
- (d) Perform load balancing across nodes (see Sect. 3.2.2).
- (e) Perform load balancing within nodes (see Sect. 3.2.2).

2. Send termination signal to worker processors.

3. Return U as the estimate of the global minimum of $f(x)$ over \mathcal{D} .

Worker Processor p

1. Initialisation

- (a) Receive workload \mathcal{L}^p from master processor.

⁹ Note that \mathcal{R} is always a finite set. As the radius is halved each time a ball is split, there can only be a finite number of radii before numerical underflow occurs.

- (b) Calculate bounds $\underline{f}(\mathcal{B})$ and $\overline{f}(\mathcal{B})$ as defined in (3.1) and (3.2), respectively, for each ball $\mathcal{B} \in \mathcal{L}^p$ and convert \mathcal{L}^p into a priority queue w.r.t. $\underline{f}(\mathcal{B})$.
 - (c) Set $U_p := \min_{\mathcal{B} \in \mathcal{L}^p} \overline{f}(\mathcal{B})$.
 - (d) Asynchronously send U_p and $|\mathcal{L}^p|$ to master processor.
 - (e) Asynchronously receive U from master processor.
2. Repeat the following procedure until termination signal is received:
- (a) Pruning: Remove from the priority queue \mathcal{L}^p balls \mathcal{B} such that $\underline{f}(\mathcal{B}) > U - \varepsilon$.
 - (b) Branching: Let \mathcal{B} be the first element in the priority queue \mathcal{L}^p .¹⁰ Split \mathcal{B} into 3^n overlapping sub-balls $\mathcal{B}_1, \dots, \mathcal{B}_{3^n}$ according to the splitting rule in Sect. 2.2 of Fowkes et al. [16] and discard any sub-balls that lie entirely outside of \mathcal{D} . Let \mathcal{R} denote the list of remaining sub-balls.
 - (c) Hashing: Generate an integer hash for each ball in \mathcal{R} and an integer hash for the radius. Send the integer hashes to master processor to see if any of the balls already exist. (Synchronised hashing only: Start bounding $f(x)$ for each ball in \mathcal{R} until the master processor sends the results of the check back). Receive an ordered integer list from the master processor that contains either 1 or 0 depending on whether each ball exists and update \mathcal{R} accordingly.
 - (d) Bounding: Calculate bounds $\underline{f}(\mathcal{B}), \overline{f}(\mathcal{B})$ according to (3.1), (3.2), for each ball $\mathcal{B} \in \mathcal{R}$ if not already bounded.
 - (e) Remove the split ball \mathcal{B} from the priority queue \mathcal{L}^p and add the list of remaining sub-balls \mathcal{R} to \mathcal{L}^p .
 - (f) Set $U_p := \min_{\mathcal{B} \in \mathcal{L}^p} \overline{f}(\mathcal{B})$.
 - (g) Load Balancing: Asynchronously send the requested number of subproblems from the current workload to the required processor(s) as instructed by the master processor and update \mathcal{L}^p accordingly. If more subproblems are requested than in the current workload, send as many as possible. Send confirmation to the master processor once the send has completed.
 - (h) Load Balancing: Asynchronously receive subproblems from other processors and update \mathcal{L}^p accordingly.
 - (i) Asynchronously send U_p and $|\mathcal{L}^p|$ to master processor.
 - (j) Asynchronously receive U from master processor.

An important performance improvement to oBB we implement is the use of a priority queue to store the subproblems. A priority queue is simply an ordered list where each element is ranked according to a specified order. In our case we order the list of balls according to the lower bound $\underline{f}(\mathcal{B})$, with the smallest lower bound included first in the list. This enables us to restructure oBB so that we do not need to find or communicate the smallest lower bound, resulting in a more efficient algorithm in standard numerical form [8].

3.2.1 Hashing

For Step 1c) of Algorithm 3.2, the master processor keeps a list of all the balls created so far and any new balls created by the worker processors are cross-checked against this list to see if they already exist. Of course, such an approach relies heavily on the ability to efficiently communicate centres and radii from the workers to the master. Sending the centre and radius

¹⁰ Note that since \mathcal{L}^p is a priority queue w.r.t. $\underline{f}(\mathcal{B})$, \mathcal{B} has the smallest lower bound $\underline{f}(\mathcal{B})$ of all balls in \mathcal{L}^p .

of each ball would be prohibitively expensive, but if we instead send an integer hash ([21], Section 6.4) of each centre and radius, this greatly decreases the cost of communication. In fact, every time a worker processor splits a ball it needs to check whether at most 3^n balls of the same radius exist. Thus, we only need to send a hash of one radius and at most 3^n balls, so $3^n + 1$ integers in total. Of course, the hashes are not guaranteed to be unique and there is a chance that the algorithm will occasionally discard a ball that does not already exist. However, such an event is extremely rare, likely to have a very small effect on the resulting minimum and can be easily corrected for by running a local solver at the end of the algorithm.

An important performance consideration is the order in which the master processor deals with the incoming hashes and we have found two different approaches to be suitable. The first approach is perhaps the most obvious, the master processes the hashes one at a time as they are received and the workers simply wait for confirmation of which balls already exist before bounding them; see *One-at-a-time Hashing* in Sect. 3.2.1 [5] for a detailed description. While this approach is suitable in situations where the balls are inexpensive to bound relative to the cost of communicating the hashes, it does not perform as well when they are not. This is because the workers tend to spend a significant amount of time waiting for a response from the master. The second approach therefore tries to address these issues by getting the master to process the hashes from all the workers in one go while the workers start bounding the balls in the background; see *Synchronous Hashing* in Sect. 3.2.1 [5] for a detailed description. This is indeed advantageous if bounding the balls is expensive relative to the communication cost. Full details on the structure and updating of the lists \mathcal{R} of radius hashes and \mathcal{C} of centre hashes, as well as our approach to hash computation are given in Sect. 3.2.1 [5].

3.2.2 Load balancing

Our load balancing scheme in Algorithm 3.2 takes into account the underlying topology of modern HPC clusters by alternately balancing across and within the underlying physical nodes of the cluster. For load balancing across processors within each node, at each load balancing step the master processor takes a snapshot of the load on the node and works out how many subproblems each processor within that node should have in order to be balanced. It then assigns the shortfall from the processor with the largest load to the one with the smallest, updates the snapshot and repeats until all processors in the node have a load that does not differ by more than 10%; see *Load Balancing across processors within a node* in Sect. 3.2.2 of Cartis et al. [5] for details.

For load balancing across nodes, at each load balancing step the master processor takes a snapshot of the overall load on each node and works out how many subproblems each node should have in order to be balanced. It then assigns a fraction of the shortfall from the processor with the largest load and distributes it as evenly as possible to all processors on the node with the smallest load. The snapshot of the total load on each node is then updated and the process repeats until all the nodes have a load that does not differ by more than 10%; see *Load Balancing across nodes* in Sect. 3.2.2 of Cartis et al. [5] for details.

3.3 Numerical results

We will now test the parallel performance of our data parallel and task parallel algorithms, namely Algorithm 3.1 and Algorithm 3.2, respectively, on the random polynomials and RBFs from Algorithm 2.3 along with RBF approximations to a selection of problems from the COCONUT benchmark [37]. We will also run the serial code, namely Algorithm 3.1 on one processor core, so that we can compare the parallel performance against the serial. In

order to do this we will calculate the speedup S_P of the parallel algorithm on P processor cores over the serial defined as

$$S_P = \frac{T_1}{T_P} \quad (3.3)$$

where T_1 is the runtime of the serial algorithm and T_P the runtime of the parallel algorithm on P processors.¹¹ The hardware used is part of the ECDF Eddie cluster where each node consists of two six-core 2.4 GHz Intel Xeon E5645 processors with 2 GB of RAM per core and the nodes communicate via Gigabit Ethernet.

3.3.1 Random polynomials and RBFs

Let us begin by looking at the parallel performance of our data and task parallel algorithms on the random polynomials and RBFs from Sect. 2.3. To this end, we will run a parallel Python-based MPI implementation of both Algorithm 3.1 and Algorithm 3.2 with one-at-a-time hashing to an absolute tolerance of 10^{-6} , i.e. we set $\varepsilon = 10^{-6}$ in step 2a of Algorithm 3.1 and in step 2a of Algorithm 3.2. As the first two series of problems described in Sect. 2.3 are very fast to solve in serial, there is little to be gained from running them in parallel and we will therefore focus on series three only. We will test the same ten realisations of series three from Sect. 2.3 in both serial and parallel for 12, 24, 36, 48 and 60 processor cores. Based upon the results of Sect. 2.3 we will use cubic underestimators (2.12) with the second order tensor Gershgorin estimation approach given in (2.15) for the random polynomials and the second order tensor norm approach given in (2.16) for the random RBFs.

Let us first consider the parallel performance of our data parallel algorithm, Algorithm 3.1. In the top of Fig. 5 we can see the average speedup with confidence intervals for Algorithm 3.1 over random polynomial and RBF series three as the number of processor cores is increased. We can see that we get rather poor sublinear speedup for both with a maximum of 17 times average speedup for the random polynomials and 24 times average speedup for the random RBFs on 60 processor cores. The reason for the better performance of the random polynomials is that calculating the elementwise lower and upper bounds is more demanding as it uses a more sophisticated interval arithmetic approach (see Section 6.2 of Fowkes et al. [16], for details) and therefore the worker processors are better utilised.

Let us now look at the performance of our task parallel algorithm, Algorithm 3.2 with one-at-a-time hashing. We have found that one-at-a-time hashing significantly outperforms synchronised hashing for both random polynomials and RBFs because the subproblems are inexpensive to solve relative to the cost of communicating the hashes. We can immediately see from the bottom of Fig. 5 that our task parallel algorithm performs significantly better than the data parallel algorithm. Random RBFs exhibit the best performance with superlinear speedup until 48 processor cores with a maximum of 55 times average speedup before levelling off. Random polynomials do not perform as well with superlinear speedup until 24 processor cores with a maximum of 36 times average speedup before dipping slightly, nonetheless the performance is still much better than that achieved with the data parallel algorithm. The poorer performance of the random polynomial problems is due to the fact that they are very quick to solve, taking only around four hundred seconds on 12 processor cores (see Table

¹¹ Note that another approach to test the efficiency of parallelism is to calculate the redundancy defined as $(F_P - F_1)/F_P$ when $F_P > F_1$ and 0 otherwise, where F_1 is the number of function evaluations of the serial algorithm and F_P the number of function evaluations of the parallel algorithm on P processors (see p. 324 of Strongin and Sergeyev [40]).

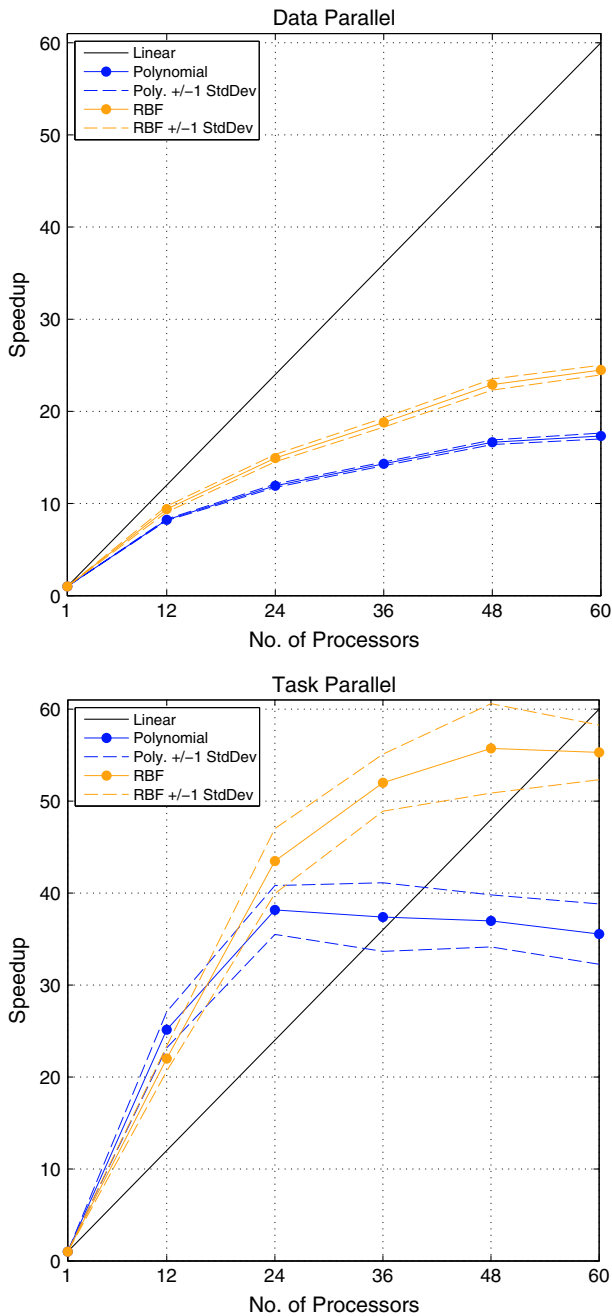


Fig. 5 Average speedup (3.3) with confidence intervals for Algorithm 3.1 (Data Parallel, *top*) and Algorithm 3.2 (Task Parallel, *bottom*) over random polynomial and RBF series 3

A.2, Appendix A [5]), which coupled with the fact that the subproblems themselves are fast to solve, leaves little scope for improvement by adding additional processor cores.

3.3.2 COCONUT benchmark

For a more thorough numerical evaluation we will test the parallel performance of our data parallel and task parallel algorithms on radial basis function approximations to a selection of 31 problems¹² from the COCONUT benchmark whose dimension varies from 4 to 6 (see [37], for details of the benchmark). Table A.3 in Appendix A of Cartis et al. [5] gives a brief overview of the test functions we have approximated. We chose to approximate the COCONUT problems using RBFs as they enable us to calculate the tensor bounds (2.14) used in the estimation approach for the lower bound (3.1) efficiently using a more accurate interval arithmetic type approach (see Section 6.2 of Fowkes et al. [16]). As before, we will use cubic underestimators (2.12) with the second order tensor norm approach given in (2.16).

The RBF approximations are fitted to a maximin Latin Hypercube sample of $10n$ scattered sample points in \mathbb{R}^n and use the cubic spline objective function we have previously used for the random RBFs in (2.19) with a weighted norm (see Chapters 3 and 4 of Fowkes [15], for details). Once again, we will run a parallel Python-based MPI implementation of both Algorithm 3.1 and Algorithm 3.2 but this time with synchronous hashing. This is because we have found that synchronous hashing leads to significantly better performance for our approximation to the COCONUT benchmark since in general the subproblems are expensive to bound relative to the cost of communicating the hashes. So that we can test both easier and harder problems we will run each problem to the absolute tolerance it achieved in 12 h on the serial code. We will test all 31 problems in both serial and parallel for 12, 24, 36, 48 and 60 processor cores. We will use the tensor norm approach given Theorem 2.4 (2.16) as we have shown in Sect. 2.3 that it performs better for RBF approximations.

As before, let us start by looking at the performance of our data parallel algorithm, Algorithm 3.1, on our approximation to the COCONUT benchmark. In the top of Fig. 6 we can see that the performance is in fact very poor with an average speedup of around three times all the way through from 12 to 60 processors. This is very disappointing but not unexpected as bounding subproblems is not where the majority of the work in the algorithm takes place but it is in exploring the branch and bound tree.

Looking at the performance of our task parallel algorithm, Algorithm 3.2 with synchronous hashing in the bottom of Fig. 6, we can see significantly better speedup. In fact, we are able to achieve superlinear speedup on average up to 36 processor cores, past which the speedup continues to increase, albeit remaining slightly sublinear, up to a maximum of 52 times average speedup.

In conclusion, we can clearly see from the numerical results that a task parallel approach leads to a very efficient parallel algorithm on average which exhibits good speedup. The data parallel algorithm on the other hand performs rather poorly, especially on our approximation to the COCONUT test set. This is due to the fact that the subproblem bounding which is parallelised in the data parallel algorithm does not account for majority of the computational work in exploring the branch and bound tree. Overall, we have shown that it is indeed possible to devise an efficient parallel overlapping branch and bound algorithm albeit after overcoming some underlying difficulties.

¹² Note that the majority of problems in the COCONUT benchmark have nonlinear constraints that our algorithms cannot handle at present. This rather limited the number of problems we could actually test.

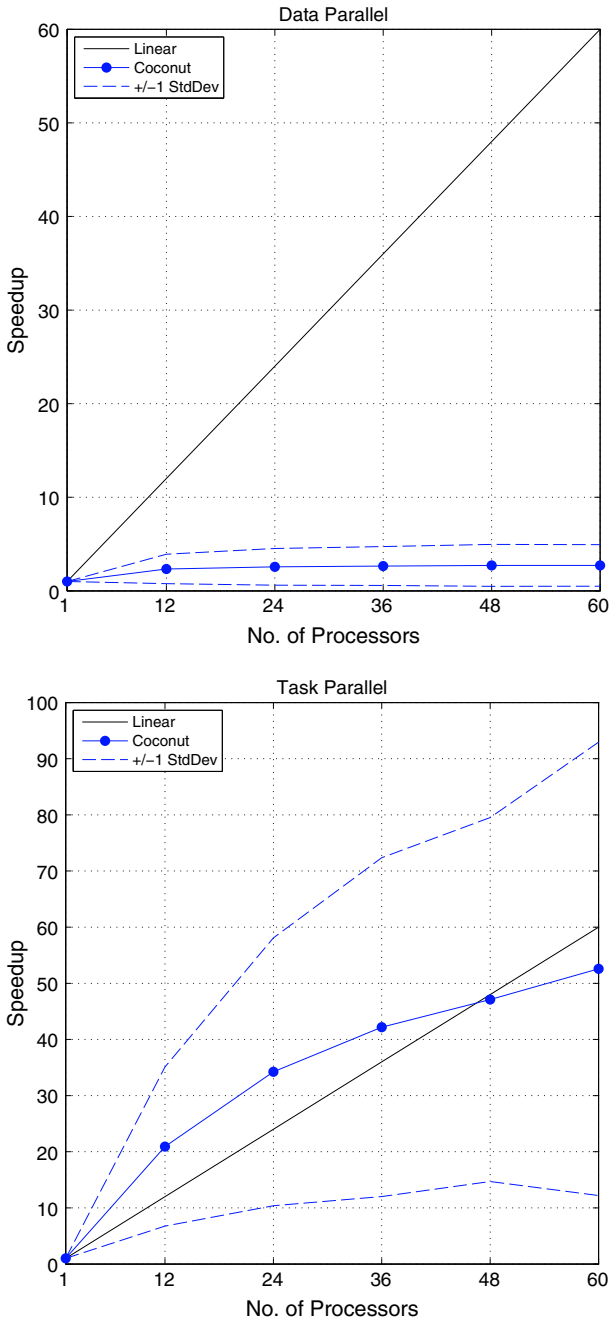


Fig. 6 Average speedup (3.3) with confidence intervals for Algorithm 3.1 (Data Parallel, *top*) and Algorithm 3.2 (Task Parallel, *bottom*) over RBF approximations to selected functions from the COCONUT test set

4 Conclusions

We have presented branching and bounding improvements for global optimization algorithms with Lipschitz continuity properties and implemented our findings by improving a recent serial branch and bound algorithm presented in Fowkes et al. [16]. We have shown that it is possible to significantly improve upon the bounding strategies used in Lipschitz based global optimization algorithms by drawing upon a variety of both existing and novel bounds. Our numerical results indicate that no single bound is optimal across all types of objective function, although our novel second order bounds exhibit the best performance in general. As these bounds are inexpensive to calculate for small to medium-scale problems compared with the cost of the rest of the algorithm it is feasible to implement all of them in a branch and bound algorithm and adaptively choose the best bound at runtime. Clearly, the latter would be the best way to maximise the efficiency of Lipschitz based branch and bound algorithms.

Our second avenue of investigation considered improving the branching used in a Lipschitz based global optimization algorithm through the use of parallelism. We investigated two standard parallel programming paradigms, namely data parallelism and task parallelism. We found that our data parallel approach which focused on parallelising the bounding operations within the algorithm performed poorly. However, our task parallel approach which focused on parallelising the branch and bound tree itself and proved to be a real challenge to realise, proved very successful once implemented and exhibited excellent average speedup on a large number of varied test problems. Our use of hashing within the task parallel algorithm was essential to obtaining good performance and we identified two main strategies for processing the hashes, namely one-at-a-time and synchronous hashing. Once again, adaptively choosing between these two strategies in the algorithm would maximise its efficiency over a large variety of problems.

A critical challenge remains: scaling up the problem dimension so that we can solve problems of greater practical interest. Scaling up the code to larger parallel machines seems like an immediate solution but may not yield results as good as one might expect due to the increased communication overhead this would bring. In the context of our approach, a better remedy perhaps lies in finding more efficient coverings that would still allow us to use non-convex bounding procedures within the algorithm.

Acknowledgments The work of the first and second authors was supported by EPSRC grants EP/I028854/1 and NAIS EP/G036136/1 and the work of the third author by EP/I013067/1. We are also grateful to NAIS for funding computing time.

References

1. Alba, E., Almeida, F., Blesa, M., Cotta, C., Díaz, M., Dorta, I., Gabarró, J., León, C., Luque, G., Petit, J., Rodríguez, C., Rojas, A., Xhafa, F.: Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. *Parallel Comput.* **32**(5–6), 415–440 (2006). doi:[10.1016/j.parco.2006.06.007](https://doi.org/10.1016/j.parco.2006.06.007)
2. Ananth, G.Y., Kumar, V., Pardalos, P.M.: Parallel processing of discrete optimization problems. In: Kent, A., Williams, J. (eds.) *Encyclopedia of Microcomputers*, vol. 13, pp. 129–147, Dekker (1993). <http://books.google.co.uk/books?id=Rx3hqGdXcooC>
3. Baritomba, W., Cutler, A.: Accelerations for global optimization covering methods using second derivatives. *J. Glob. Optim.* **4**(3), 329–341 (1994). doi:[10.1007/BF01098365](https://doi.org/10.1007/BF01098365)
4. Breiman, L., Cutler, A.: A deterministic algorithm for global optimization. *Math. Program.* **58**(1–3), 179–199 (1993). doi:[10.1007/BF01581266](https://doi.org/10.1007/BF01581266)
5. Cartis, C., Fowkes, J.M., Gould, N.I.M.: Branching and bounding improvements for global optimization algorithms with Lipschitz continuity properties. Tech. rep., Optimization. http://www.optimization-online.org/DB_HTML/2013/06/3914.html (2013)

6. Casado, L.G., Martinez, J.A., Garcia, I., Hendrix, E.M.T.: Branch-and-bound interval global optimization on shared memory multiprocessors. *Optim. Methods Softw.* **23**(5), 689–701 (2008). doi:[10.1080/10556780802086300](https://doi.org/10.1080/10556780802086300)
7. Conn, A.R., Gould, N.I.M., Toint P.L.: Trust region methods. MPS-SIAM series on optimization, SIAM. <http://books.google.co.uk/books?id=5kNC4fqssYQC> (2000)
8. Crainic, T.G., Le Cun, B., Roucairol, C.: Parallel branch-and-bound algorithms. In: Talbi, E. (ed.) *Parallel Combinatorial Optimization*, Wiley Series on Parallel and Distributed Computing, pp. 1–28. Wiley, New York. http://books.google.co.uk/books?id=rYtuk_sm23UC (2006)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. *Math. Program.* **91**(2), 201–213 (2002). doi:[10.1007/s101070100263](https://doi.org/10.1007/s101070100263)
10. Evtushenko, Y.G.: Numerical methods for finding global extrema (case of a non-uniform mesh). *USSR Comput. Math. Math. Phys.* **11**(6), 38–54 (1971)
11. Evtushenko, Y.G., Posypkin, M.A.: An application of the nonuniform covering method to global optimization of mixed integer nonlinear problems. *Comput. Math. Math. Phys.* **51**(8), 1286–1298 (2011). doi:[10.1134/S0965542511080082](https://doi.org/10.1134/S0965542511080082)
12. Evtushenko, Y.G., Posypkin, M.A.: A deterministic approach to global box-constrained optimization. *Optim. Lett.* **7**(4), 819–829 (2013). doi:[10.1007/s11590-012-0452-1](https://doi.org/10.1007/s11590-012-0452-1)
13. Evtushenko, Y.G., Posypkin, M.A., Sigal, I.: A framework for parallel large-scale global optimization. *Comput. Sci. Res. Dev.* **23**(3–4), 211–215 (2009). doi:[10.1007/s00450-009-0083-7](https://doi.org/10.1007/s00450-009-0083-7)
14. Floudas, C.: *Deterministic Global Optimization: Theory, Methods and Applications*. Nonconvex Optimization and Its Applications. Springer, Berlin. <http://books.google.co.uk/books?id=qZSpq27TsOeC> (1999)
15. Fowkes, J.M.: *Bayesian Numerical Analysis: Global Optimization and Other Applications*. PhD thesis, Mathematical Institute, University of Oxford. <http://ora.ox.ac.uk/objects/uuid:ab268fe7-f757-459e-b1fe-a4a9083c1cba> (2012)
16. Fowkes, J.M., Gould, N.I.M., Farmer, C.L.: A branch and bound algorithm for the global optimization of Hessian Lipschitz continuous functions. *J. Glob. Optim.* **56**(4), 1791–1815 (2013). doi:[10.1007/s10898-012-9937-9](https://doi.org/10.1007/s10898-012-9937-9)
17. Gaviano, M., Lera, D.: A global minimization algorithm for Lipschitz functions. *Optim. Lett.* **2**(1), 1–13 (2008). doi:[10.1007/s11590-006-0036-z](https://doi.org/10.1007/s11590-006-0036-z)
18. Gendron, B., Crainic, T.G.: Parallel branch-and-bound algorithms: survey and synthesis. *Oper. Res.* **42**(6), 1042–1066 (1994). doi:[10.1287/opre.42.6.1042](https://doi.org/10.1287/opre.42.6.1042)
19. Grishagin, V.A.: Operating characteristics of some global search algorithms. *Probl. Stoch. Search* **7**, 198–206 (1978). (In Russian)
20. Horst, R., Tuy, H.: *Global Optimization: Deterministic Approaches*. Springer, Berlin. <http://books.google.co.uk/books?id=usFjGFvuBDEC> (1996)
21. Knuth, D.: *The Art of Computer Programming: Sorting and Searching*, the Art of Computer Programming, vol. 3. Addison-Wesley. <http://books.google.co.uk/books?id=ePzuAAAAMAAJ> (1998)
22. Kolda, T., Mayo, J.: Shifted power method for computing tensor eigenpairs. *SIAM J. Matrix Anal. Appl.* **32**(4), 1095–1124 (2011). doi:[10.1137/100801482](https://doi.org/10.1137/100801482)
23. Kreinovich, V., Kearfott, R.: Beyond convex? Global optimization is feasible only for convex objective functions: a theorem. *J. Glob. Optim.* **33**(4), 617–624 (2005). doi:[10.1007/s10898-004-2120-1](https://doi.org/10.1007/s10898-004-2120-1)
24. Kvasov, D.E., Sergeyev, Y.D.: A univariate global search working with a set of Lipschitz constants for the first derivative. *Optim. Lett.* **3**(2), 303–318 (2009). doi:[10.1007/s11590-008-0110-9](https://doi.org/10.1007/s11590-008-0110-9)
25. Kvasov, D.E., Sergeyev, Y.D.: Lipschitz gradients for global optimization in a one-point-based partitioning scheme. *J. Comput. Appl. Math.* **236**(16), 4042–4054 (2012). doi:[10.1016/j.cam.2012.02.020](https://doi.org/10.1016/j.cam.2012.02.020)
26. Kvasov, D.E., Sergeyev, Y.D.: Univariate geometric Lipschitz global optimization algorithms. *Numer. Algebr. Control Optim.* **2**(1), 69–90 (2012). doi:[10.3934/naco.2012.2.69](https://doi.org/10.3934/naco.2012.2.69)
27. Lera, D., Sergeyev, Y.D.: Acceleration of univariate global optimization algorithms working with Lipschitz functions and Lipschitz first derivatives. *SIAM J. Optim.* **23**(1), 508–529 (2013). doi:[10.1137/110859129](https://doi.org/10.1137/110859129)
28. Lim, L.H.: Singular values and eigenvalues of tensors: a variational approach. In: *Computational Advances in Multi-Sensor Adaptive Processing*, 2005 1st IEEE International Workshop on, pp. 129–132 (2005). doi:[10.1109/CAMAP.2005.1574201](https://doi.org/10.1109/CAMAP.2005.1574201)
29. Neumaier, A.: Complete search in continuous global optimization and constraint satisfaction. *Acta Numer.* **13**, 271–369 (2004). doi:[10.1017/S0962492904000194](https://doi.org/10.1017/S0962492904000194)
30. Pardalos, P.M., Horst, R., Thoai, N.V.: *Introduction to Global Optimization*, Nonconvex Optimization and its Applications, vol. 3. Springer, Berlin. <http://www.springer.com/mathematics/book/978-0-7923-3556-6> (1995)

31. Paulavičius, R., Žilinskas, J., Grothey, A.: Parallel branch and bound for global optimization with combination of Lipschitz bounds. *Optim. Methods Softw.* **26**(3), 487–498 (2011). doi:[10.1080/10556788.2010.551537](https://doi.org/10.1080/10556788.2010.551537)
32. Pintér, J.D.: *Global Optimization in Action, Nonconvex Optimization and its Applications*, vol. 6. Springer, Berlin. <http://www.springer.com/mathematics/book/978-0-7923-3757-7> (1996)
33. Piyavskii, S.A.: An algorithm for finding the absolute extremum of a function. *USSR Comput. Math. Math. Phys.* **12**(4), 57–67 (1972). doi:[10.1016/0041-5553\(72\)90115-2](https://doi.org/10.1016/0041-5553(72)90115-2)
34. Qi, L.: Eigenvalues of a real supersymmetric tensor. *J. Symb. Comput.* **40**(6), 1302–1324 (2005). doi:[10.1016/j.jsc.2005.05.007](https://doi.org/10.1016/j.jsc.2005.05.007). <http://www.sciencedirect.com/science/article/pii/S0747717105000817>
35. Sergeyev, Y.D.: Global one-dimensional optimization using smooth auxiliary functions. *Math. Program.* **81**(1), 127–146 (1998). doi:[10.1007/BF01584848](https://doi.org/10.1007/BF01584848)
36. Sergeyev, Y.D., Strongin, R.G., Lera, D.: *Introduction to Global Optimization Exploiting Space-Filling Curves*. Springer Briefs in Optimization, Springer, Berlin. <http://books.google.co.uk/books?id=IqyYnAEACAAJ> (2013)
37. Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.H., Nguyen, T.V.: Benchmarking global optimization and constraint satisfaction codes. In: Blik, C., Jermann, C., Neumaier, A. (eds.) *Global Optimization and Constraint Satisfaction*, Lecture Notes in Computer Science, vol. 2861, pp. 211–222. Springer, Berlin, Heidelberg (2003). doi:[10.1007/978-3-540-39901-8_16](https://doi.org/10.1007/978-3-540-39901-8_16)
38. Shubert, B.: A sequential method seeking the global maximum of a function. *SIAM J. Numer. Anal.* **9**(3), 379–388 (1972). doi:[10.1137/0709036](https://doi.org/10.1137/0709036)
39. Stephens, C.P., Baritomba, W.: Global optimization requires global information. *J. Optim. Theory Appl.* **96**(3), 575–588 (1998). doi:[10.1023/A:1022612511618](https://doi.org/10.1023/A:1022612511618)
40. Strongin, R.G., Sergeyev, Y.D.: *Global Optimization with Non-convex Constraints: Sequential and Parallel Algorithms*. Nonconvex Optimization and its Applications. Springer, Berlin. http://books.google.co.uk/books?id=xh_GF9Dor3AC (2000)
41. Zhang, X., Qi, L., Ye, Y.: The cubic spherical optimization problems. *Math. Comput.* **81**(279), 1513–1525 (2012). doi:[10.1090/S0025-5718-2012-02577-4](https://doi.org/10.1090/S0025-5718-2012-02577-4)