BRUNO ALEXANDRE DA ANUNCIAÇÃO BAPTISTA

Bachelor in Mathematics

INCORPORATING RADIAL BASIS FUNCTIONS IN GLOBAL AND LOCAL DIRECT SEARCH

MASTER IN MATHEMATICS AND APPLICATIONS

NOVA University Lisbon
September, 2022

INCORPORATING RADIAL BASIS FUNCTIONS IN GLOBAL AND LOCAL DIRECT SEARCH

BRUNO ALEXANDRE DA ANUNCIAÇÃO BAPTISTA

Bachelor in Mathematics

Adviser: Ana Luísa Custódio
Associate Professor, NOVA University Lisbon

Co-adviser: Carmo P. Brás
Associate Professor, NOVA University Lisbon

MASTER IN MATHEMATICS AND APPLICATIONS

NOVA University Lisbon
September, 2022

Acknowledgements

I would like to thank my advisors Professor Ana Luísa Custódio and Professor Carmo P. Brás. They helped immensely, by guiding me through this work, and lending a helping hand whenever progress was not being made. Without their tremendous efforts, this work would have not been possible. It is entirely due to them that I can say I am proud of the quality of the proposed algorithm.

I would also like to thank FCT - Fundação para a Ciência e Tecnologia for funding and support of this work under the scope of project PTDC/MAT-APL/28400/2017, as well as NOVA Math (Center for Mathematics and Applications) for access to equipment and logistic facilities.

Finally, I would like to thank my family for all of the emotional support provided. They played a crucial role in making sure I was in the right mindset for this work.

Abstract

GLODS is a global derivative-free optimization algorithm, relying on local directional direct search, aided by a clever multistart strategy that does not conduct all the lines of search until the end. In 2015, time of the first release of the corresponding solver, GLODS was shown to be competitive when compared to state-of-the-art algorithms, such as MCS or DIRECT.

GLODS resorts to sampling techniques to look for minima on a global scale, not taking advantage of the information gathered in previous iterations. As such, the main goal of this work is to replace the pseudo-random sampling approach, used by GLODS to initialize new lines of search, by the minimization of global models of the objective function, defined using radial basis functions, and computed using the points previously evaluated by the algorithm. This should allow a better placement of the starting points for new local lines of search, and, in turn, significantly increase the numerical performance of the algorithm.

Naturally, incorporating radial basis functions in GLODS poses new challenges. In this work, we will address questions such as which radial basis functions to use, which points should be selected to compute them, how to minimize these functions, and how to take advantage of their minima in the execution of the algorithm.

The new version of GLODS, incorporating radial basis functions, was calibrated to its best numerical performance, and then compared against other state-of-the-art solvers, such as MCS, DIRECT, MATSuMoTo, and ZOOpt. The results obtained are strongly positive. The new algorithm clearly outperforms its previous version, and is competitive with the other solvers tested.

Finally, parallel strategies were implemented and tested. Results showed that it is very beneficial to evaluate multiple points simultaneously, for objective functions whose evaluation time is as low as 0.1 seconds. The proposed algorithm, called BoostGLODS, is a cutting-edge, powerful and efficient parallel global derivative-free optimization algorithm.

Keywords: Global optimization, Derivative-free optimization, Radial basis functions, Surrogate models, Direct search methods, Pattern search methods, Nonsmooth optimization

Resumo

O algoritmo GLODS é um método de otimização global sem recurso a derivadas, baseado em procura direta direcional, aliada a uma estratégia de reinicialização inteligente, que não leva todas as linhas de procura até ao final. Em 2015, ano em que foi disponibilizada a primeira versão do correspondente código, o algoritmo GLODS mostrou-se competitivo com outros algoritmos do estado da arte, como sejam o MCS e o DIRECT.

O algoritmo GLODS usa técnicas de amostragem pseudo-aleatórias para procurar mínimos numa escala global, não tirando partido da informação adquirida em iterações anteriores. Assim, o objetivo principal deste trabalho é substituir as estratégias de amostragem pseudo-aleatória, usadas pelo GLODS para inicializar novas linhas de pesquisa, pela minimização de modelos globais da função objetivo, definidos à custa de funções de base radial, que são construídos usando pontos que o algoritmo já avaliou em iterações anteriores. Esta substituição deverá permitir ao GLODS um melhor posicionamento dos pontos para inicialização de procuras locais, o que, por sua vez, deverá levar a um aumento significativo do desempenho numérico do algoritmo.

Naturalmente, incorporar funções de base radial no GLODS traz desafios adicionais. Neste trabalho, iremos responder a perguntas como que funções de base radial usar, que pontos selecionar para as construir, como minimizar estas funções e como incorporar essa informação na execução do algoritmo.

A nova versão do GLODS, que já contempla o uso de funções de base radial, foi calibrada com vista a ter o melhor desempenho numérico. Posteriormente, foi comparada com outros algoritmos do estado da arte, como sejam o MCS, o DIRECT, o MATSuMoTo, e o ZOOpt. Os resultados obtidos são muito positivos. O novo algoritmo mostra um desempenho claramente superior à sua anterior versão, e é competitivo com os restantes algoritmos testados.

Finalmente, foram implementadas e testadas estratégias de paralelismo. Os resultados mostram que é bastante benéfico avaliar vários pontos em simultâneo, para funções objetivo cujo tempo de avaliação é tão baixo como 0.1 segundos. O algoritmo proposto, designado por BoostGLODS, é um algoritmo paralelo de otimização global sem recurso a derivadas, com estratégias de ponta, poderoso e eficiente.

Palavras-chave: Otimização global, Otimização sem derivadas, Funções de base radial, Modelos sub-rogados, Métodos de procura direta, Métodos de procura em padrão, Otimização não suave

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Context and motivation

In global optimization, the goal is to find the best possible value for a given objective function, respecting the existing constraints. Problems of this kind can be simply treated as minimization problems, since any maximization problem can be formulated as a minimization one, by considering the symmetric of its objective function.

For unconstrained convex functions, a local minimum can be found using one of the classical local optimization methods, such as the Gradient Descent algorithm [36], when in presence of derivatives, or directional direct search methods [13], otherwise. In this case, as the objective function is convex, there can only be one extremum point. Therefore, the local minimum found is also the global minimum. However, for non-convex functions, many local minima may exist, which makes it harder for an algorithm to identify the global minimum or to guarantee that it has been identified. For this reason, global optimization is a scientific domain that often proves itself much more challenging than its local counterpart.

In this work, the objective functions for which we want to compute the global minimum do not have derivatives available. This is because the expression defining the objective function is unavailable, which is commonly referred to as *Black-box Optimization*, the objective function is non-smooth, or a significant amount of noise is present in its evaluation. Thus, our task is even tougher, as we cannot rely on derivative-based methods. Despite this substantial increase in difficulty, these problems arise in multiple areas of study, as is the case with the design of hybrid electric vehicles [18], reinforcement learning in robotics [28], molecular conformal optimization problems [4], acoustics [34] or multidisciplinary design optimization [38]. As such, developing algorithms for global derivative-free optimization is a very relevant study area.

1.2 State of the art

MCS [20] and DIRECT [24] are two state-of-the-art solvers suited for global bound-constrained derivative-free optimization, inspired by branch and bound algorithms [27] in the area of combinatorial optimization. Both algorithms rely on successively partitioning the feasible region, in an attempt to identify and narrow down the regions that yield the lowest function values. Areas identified as promising are explored further, while uninteresting ones are left behind, largely unexplored. The main difference between the two algorithms lies in the way this exploration takes place. DIRECT simply divides the promising regions into smaller ones, whereas in MCS a local optimization procedure is conducted by computing local models of the objective function. In addition to the global minimum, MCS is also capable of identifying some local minima. Despite having been released more than 20 years ago, both solvers are still renowned global derivative-free optimization algorithms, continuing to be used to benchmark new methods.

As an alternative to partitioning the feasible region, one could start multiple local search lines from different points scattered throughout the feasible region. This is the idea behind multistart strategies, which are also part of the Global and Local Optimization using Direct Search (GLODS) algorithm [15]. Although simple at its core, this approach requires careful thinking of how the starting points should be selected, as well as how many points should be considered - having too many or not spreading them enough might have consequences in the efficiency of the algorithm, whilst not significantly improving the quality of the minima found, as several local lines of search might converge to the same point, wasting precious budget. On the other hand, choosing too few starting points might lead to a very scarce search in the feasible region, possibly leaving interesting areas unexplored. GLODS tries to address these questions by granting each point evaluated a comparison radius, which is then used to merge different local lines of search when they come too close to each other, preventing unnecessary function evaluations and significantly speeding up the process of finding the global minimum, as well as leaving more budget to further explore the feasible region.

Another possible approach to global derivative-free optimization problems consists in the use of the so-called *surrogate functions* - functions that are simultaneously cheap to compute and capable of mimicking the behavior observed in the original objective function, albeit with much less detail. One way of using these functions is to alternate between building a model of the objective function, using points previously evaluated and the surrogate function of choice, and minimizing that model to then evaluate the minima found on the real objective function, until a stopping criterion is met. Naturally, it will be much less computationally demanding to minimize models based on surrogate functions than the original objective function. This is the idea behind MATSuMoTo [31], where *radial basis functions* [9, 49] were chosen as surrogate functions, allowing an efficient exploration of the feasible region. ORBIT [50] also resorts to using radial basis functions as surrogates for the real, expensive function, but its algorithmic structure is different from the one previously described, as it uses a *trust-region* [12] approach.

Although widely used and with interesting results, the quality of surrogate models is highly dependent on the sample set considered. Acquisition functions [43] can help with the selection of adequate sample sets. Building upon an already existing surrogate model, they promote the exploration of areas of the feasible region that have not yet been explored. Models based on acquisition functions are less likely to stagnate early on harder problems than ones purely based on surrogate functions.

Another state-of-the-art global derivative-free optimization algorithm is SRacos [19], used by the ZOOpt [29] optimization toolbox. SRacos is a solver that tries to find the global minima of an objective function via *classification* techniques [51]. This means that its main goal is to build a model capable of classifying solutions as either *good* or *bad*. In SRacos, at each iteration, the best solution is classified as good, and the remaining ones as bad. Sampling from areas outside the vicinity of bad solutions is then preferred and the process repeats itself until no more function evaluations are allowed. Only the best point found so far carries over to the next iteration. When the solver stops, the feasible region is expected to be moderately well explored, and a good solution is expected to have been identified.

1.3 Objectives and thesis organization

The main objective of this thesis is to improve the numerical performance of the GLODS algorithm by using radial basis functions. As described in Section 1.2, GLODS uses an adapted multistart strategy, where new local searches are initialized but not always conducted until the end. However, these new initializations are performed with sampling techniques, not taking advantage of the information gathered by the algorithm in previous iterations. The idea to be explored in this work is how to take advantage of this information, by incorporating surrogate models in GLODS, leading to an improvement of the numerical performance of the code. By using previously evaluated points to compute a rough model of how the objective function behaves globally, instead of looking randomly inside the feasible region, GLODS should be able to better place new starting points for local searches, and, as a consequence, reach the global minimum faster, or find additional local minima.

The rest of this work is organized as follows. In Chapter 2, a review of the original GLODS algorithm is presented, including its convergence analysis (see Section 2.2). Then, in Chapter 3, we introduce radial basis functions. This includes the considerations necessary to handle them in any solver (see Section 3.3). Afterwards, in Chapter 4, the new search step is proposed, and additional considerations pertaining to incorporating surrogate models in GLODS are detailed. Then, the study conducted to calibrate the new algorithm to its best numerical performance begins. Firstly, in Chapter 5, the tools used to assess the performance of solvers are presented. Then, in Chapter 6, multiple versions of the new algorithm are tested against each other and ruled out, until only the final best version remains. Finally, in Chapter 7, the best version is compared against other state-of-the-art solvers. Two additional chapters remain. In Chapter 8, considerations regarding parallel strategies are tested, and their results presented. Finally, in Chapter 9, we draw conclusions from the results obtained, and some interesting alternatives to some of the decisions taken are mentioned.

2 The original GLODS

In this chapter, we will detail the main algorithmic structure of GLODS, as well as its algorithmic variants and which of these came out as the top performers in the tests conducted at the time the solver was first released, and are now used as GLODS' default settings. Some important theoretical properties regarding the convergence of the algorithm will also be stated.

2.1 Main algorithmic structure

GLODS is a global derivative-free optimization algorithm, suited for bound constrained problems, heavily relying on local directional direct search [13] aided by a clever multistart strategy. Like any directional direct search method, each iteration of GLODS is organized in a search step and/or a poll step. The goal is to be able to thoroughly explore promising areas of the feasible region using local search - the poll step -, whilst being able to identify these areas using multistart strategies - the search step.

Multistart is applied to GLODS in a simple way: in the first iteration, a few lines of search are initialized. Then, as the algorithm converges to local minima, new lines are initialized in the feasible region, but not taking advantage of the information collected during the optimization procedure. Additionally, when two lines are sufficiently close, they are merged and only one of them is further explored. This allows GLODS to be much more efficient than the classical multistart strategies.

2.1.1 The list

Throughout GLODS' execution, a list of points is kept, storing all the relevant points evaluated so far. For every point x, the list will also keep some important information about it:

1. its corresponding value in the objective function f, $f(x)$;

2. its corresponding step-size parameter α, which will dictate the distance between itself and the points that will be evaluated should x be selected for local exploration

at the poll step;

3. its comparison radius r, used to measure its closeness to other points already evaluated and stored, providing a way to merge local lines of search that come too close to each other;

4. its active or inactive state, which indicates whether x is the best point of its local line of search.

Points can be added to the list under a few conditions. A schematic description can be found in Algorithm 1. In all cases, if a newly evaluated point x_{new} is added to the list, all active points comparable to it with worse function value change their state to inactive (line 9 in Algorithm 1). If x_{new} is not comparable with any point already in the list, meaning it does not fall inside the comparison radius of any of the points in the list (line 1), then it is a point in a completely unexplored area of the domain. As such, x_{new} will be added to the list as an active point (line 2). If x_{new} is comparable to an active point ($i_{dom} > 0$) and every point comparable to x_{new} has a worse objective function value than x_{new} ($p_{dom} = 0$), then x_{new} will be added to the list as an active point. If x_{new} is comparable both to an active point and at least another point in the list, and presents a better objective function value than the active point, but a worse objective function value than the other point, then x_{new} will be added to the list as an inactive point ($i_{dom} > 0 \wedge p_{dom} = 1$).

When deciding if a point has a better objective function value, a forcing function ρ can be used (lines 6 and 13). A forcing function ρ [26] is a positive, continuous, non-decreasing function that satisfies $\rho(t)/t \to 0$ as t decreases to 0. Forcing functions are useful because they force the algorithm only to accept new points when there is a *sufficient decrease* of the objective function value, making the algorithm more robust when there is noise in function evaluation. Typical examples of forcing functions are $\rho(t) = t^{1+a}$, where $a > 0$. Alternatively, a *simple decrease* approach could be considered, where $\rho(t) = 0, \forall t \in \mathbb{R}$. In this case, in addition to the conditions described before, x_{new} may also be added to the list as an active point when it presents a better objective function value than any active or inactive point it is comparable to ($i_{comp} > 0 \wedge p_{dom} = 0$).

Algorithm 1: Adding a point x to the list L.

```
Input: $x$ - point to be added to the list;
       $L$ - list of points currently stored;
       $\alpha_0$ - $\alpha$ used in the initialization of the algorithm;
       $r_0$ - $r$ used in the initialization of the algorithm;
       $f$ - objective function;
       $\rho$ - a forcing function or the null function;
       $s_{poll}$ - 1 if called from the poll step, 0 otherwise;
       $\alpha_{poll}$ - $\alpha$ of the polling center, when applicable;
       $r_{poll}$ - $r$ of the polling center, when applicable.
if $\min_{y \in L} \|x - y\| - r_y > 0$ then
    $L = L \cup \{(x, f(x), \alpha_0, r_0, 1)\}$
else
    if $x \notin L$ then
        $\alpha_{max} = 0$, $r_{max} = 0$, $i_{dom} = 0$, $p_{dom} = 0$, $i_{comp} = 0$
        forall $(y, f(y), \alpha_y, r_y, i_y) \in L$ do
            if $\|x - y\| - r_y \leq 0$ then
                if $f(x) < f(y) - \rho(\alpha_y)$ then
                    $i_{comp} = i_{comp} + 1$
                    $i_{dom} = i_{dom} + i_y$
                    $i_y = 0$
                    if $\alpha_y > \alpha_{max}$ then
                        $\alpha_{max} = \alpha_y$
                        $r_{max} = r_y$
                    end
                else
                    if $f(y) \leq f(x) - \rho(\alpha_y)$ then
                        $p_{dom} = 1$
                    end
                end
            end
            if $p_{dom} = 0$ then
                $i_x = 1$
            end
            if $i_{dom} > 0 \vee (\rho(\cdot) \equiv 0 \wedge p_{dom} = 0 \wedge i_{comp} > 0)$ then
                if $s_{poll} = 1$ then
                    $L = L \cup \{(x, f(x), \alpha_{poll}, r_{poll}, i_x)\}$
                else
                    $L = L \cup \{(x, f(x), \alpha_{max}, r_{max}, i_x)\}$
                end
            end
        end
    end
end
```

2.1.2 Algorithmic steps

The general structure of GLODS is detailed in Figure 2.1. Each iteration of a classic direct-search method of directional type can be classified as either *successful* or *unsuccessful*. In GLODS, an additional option is possible: *merging*. A successful iteration is declared when a new active point x_{new} has been added to the list. If no new point was added, then the iteration is declared as unsuccessful. If all new points were added as inactive, the iteration is said to be a merging iteration.

2.1.2.1 The initialization step

GLODS is initialized by considering a finite set of distinct points. These points are then evaluated and Algorithm 1 is called to, hopefully, add all of them to the list. GLODS' considers the default values for α and r to be equal to 1, but these can be changed by the user. If changed, r must be at least as large as $\alpha \max_{d \in D} \|d\|$, where D is the set of poll directions considered at the current iteration, so that the points generated at the poll step are comparable with the poll center (see Section 2.1.2.2 for more information on the poll step). The solver then proceeds to the poll step or the search step, depending on the number of points in the list and the criteria chosen to perform a search step (see Section 2.1.2.3).

The initial set of points can be computed by different methods:

1. Random sampling [41], where points are randomly scattered across the feasible region;

2. Latin hypercube sampling [32], where random sampling is performed in subdomains of the feasible region, guaranteeing that randomness does not concentrate points too close to each other;

3. Points equally spaced in a line segment, connecting two opposite corners of the feasible region, jointly with the central point - GLODS' default option;

4. A user-provided list of points, useful when the user already has some knowledge of where the global minimum of the objective function might be.

Additionally, the user can also select the number of points for initialization. By default, the number of points considered is equal to the problem dimension n, when n is an odd number, and $n+1$, when n is even, in order to always include the central point in the line segment option for initialization.

2.1.2.2 The poll step

The poll step is responsible for GLODS' convergence and is also the main focus of the algorithm. For this step, it is necessary to define a set of directions D. This set must be a positive spanning set [39]. Then, the active point x_{best} with the best objective function

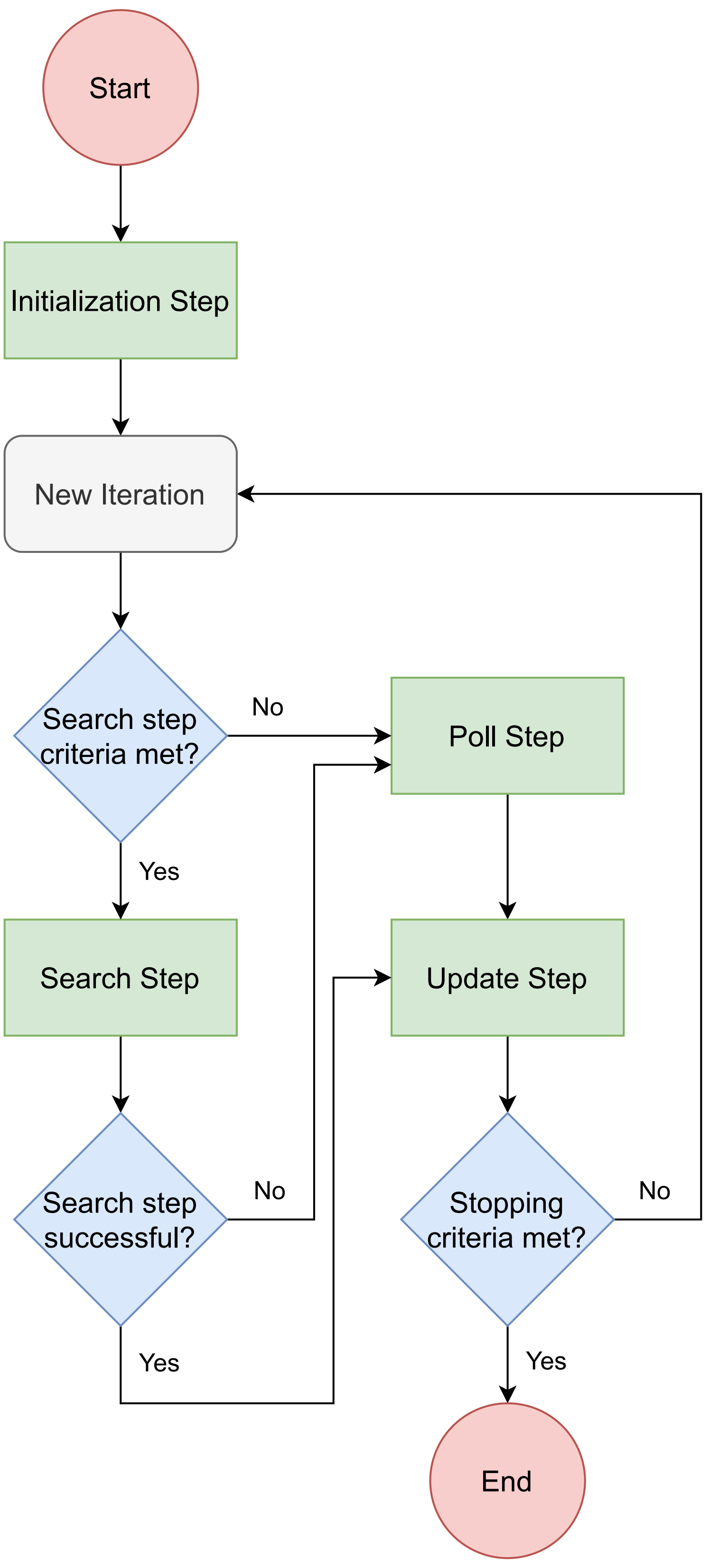

Figure 2.1: Flowchart of the general algorithmic structure of GLODS.

value is selected as a *poll center* and *polling* around it is performed, meaning that points of the form $x_{best} + \alpha_{best}d$ are evaluated, where $d \in D$. Lastly, Algorithm 1 is called to add the new points to the list.

When handling the points generated in this step, GLODS can either evaluate all of them, or until one is added to the list as active, disregarding the remaining ones. The first approach is designated as *complete* polling, whereas the second one is designated as *opportunistic* or *greedy* polling. By default, GLODS resorts to the latter.

Every iteration of GLODS performs a poll step, with the exception of when the search step is successful, meaning that a new active point was added to the list at this search step.

2.1.2.3 The search step

Since the poll step is where convergence is ensured, the search step enjoys much more freedom in its definition. However, this step must still be defined with care, as its quality directly ties with the quality of the minima found, since this step is what grants GLODS its global properties.

The search step generates new points in an attempt to cover the whole feasible region, should an infinite number of function evaluations be allowed. With this goal in mind, the search step selects the points to be evaluated based on sampling techniques:

1. Random sampling [41], like in the initialization;
2. Latin hypercube sampling [32];
3. 2^n-centers [15], a deterministic strategy developed by GLODS' authors;
4. Halton and Sobol sequences [25], two pseudo-random sampling techniques [35], meaning that they mimic the randomness of random sampling but are deterministic – given the same input multiple times, the output will always be the same.

Since the search step is not mandatory for GLODS' convergence, there is also the option of not evaluating any points in the search step, effectively disabling this step. When enabled, the search step does not need to be executed at every iteration. Strategies to choose whether to perform a search step at a given iteration are based on one of the following criteria:

1. The number of active points in the list falls below a threshold set by the user;
2. A number (also set by the user) of consecutive iterations have been unsuccessful.

By default, GLODS uses Sobol sequences when performing a search step, and a search step is performed when there is only one active point in the list. When performed, the number of points generated in the search step is equal to the dimension of the problem.

2.1.2.4 The update step

Depending on how the iteration was classified, old points in the list may see their corresponding α and r updated. Newly added points also need to have their corresponding step-size, α, and comparison radius, r, set. To do this, points are handled based on whether they were generated in the poll step or in the search step. If the points were generated in the poll step and the iteration was unsuccessful, then the point chosen for local exploration x_{poll} will have both its α_{poll} and r_{poll} reduced. If generated in the poll step and the iteration was successful or merging, x_{new} will inherit these values from x_{poll}, either keeping or increasing them. If generated in the search step, α_{new} and r_{new} are set to the values of the point with largest α found in the set of comparable points of worse objective function values. When x_{new} is not comparable to any point in the list, α_{new} and r_{new} are set to the default values used in the initialization of the algorithm. In a successful search step, the newly set values for α_{new} and r_{new} may also be kept or increased.

2.1.2.5 Stopping criteria

At the end of every iteration, a few criteria are checked, in order to decide if the algorithm should proceed to the next iteration, or if it should halt its execution. Like many other directional direct search methods, these criteria will be based on either the step-size, or on the number of function evaluations performed. The criteria considered for GLODS are:

1. The step-sizes of all active points in the list have fallen below a threshold set by the user;

2. The number of function evaluations performed has exceeded the maximum imposed by the user.

The first criterion closely relates to the convergence properties of GLODS, to be detailed in Section 2.2. GLODS identifies points as local minima by analyzing their corresponding step-size. If a point is a local minimum and is in the list, then it should be an active point (as active points are the best of their respective local lines of search). Since it is active, GLODS will eventually select it as a poll center. However, since it is a local minimum, all of its offspring will have a worse objective function value, resulting in an unsuccessful poll step. As such, its step-size will be reduced. This process repeats itself throughout the run, unless we purposely choose to stop the local search around this point by identifying it as a local minimum. The first criteria listed is met when all active points in the list have been identified as local minima. By default, GLODS considers a point a local minimum when its step-size falls below 10^{-8}.

The second criterion provides a strict limit to the number of function evaluations allowed. By default, the maximum number of function evaluations allowed is 20000. GLODS stops as soon as one of these two criteria is met.

2.2 Convergence

In this section, a few theoretical properties related to the behavior of the GLODS algorithm will be stated. However, the corresponding proofs will not be detailed in this work. These can be consulted in the original publication [15].

In a classical directional direct search method, it is guaranteed that there is always a poll center at the end of an iteration, so that the algorithm may proceed. In GLODS, this guarantee may not be obvious, as only active points can ever be selected as poll centers, and the list manipulates their active and inactive states, often turning active points inactive but never the other way around. Nevertheless, at each iteration of GLODS, there is always at least one active point in the list of points stored (see Proposition 3.1 in [15]).

The convergence analysis of GLODS relies on imposing one of two globalization strategies: integer lattices (as in Generalized Pattern Search (GPS) [5] or Mesh Adaptive Direct Search (MADS) [6]), meaning that all points generated by the algorithm lie in an implicit integer lattice, or sufficient decrease (as in Generating Set Search (GSS) [26]), meaning that ρ is not the null function, but a forcing function.

Before proceeding, a few general assumptions are necessary.

Assumption 1 *The feasible region $\Omega \subseteq \mathbb{R}^n$ is a compact set.*

Assumption 2 *The objective function f is lower-bounded in Ω.*

The first step in guaranteeing convergence in GLODS is to ensure that one sequence of points in the list generated at different iterations has the corresponding step-size parameters sequence converging to zero. In other words, the algorithm must generate a sequence of iterates that converges to some point in the feasible region.

As briefly mentioned in Section 2.1.2.2, the set of directions chosen at each iteration must be a positive spanning set. For continuously differentiable functions, a finite set of directions which satisfies appropriate integrality requirements is enough [5, 26]. Otherwise, the union of all positive spanning sets considered through all the iterations, $\mathcal{D}$, should be dense in the unit sphere. However, when considering a globalization strategy based on integer lattices, additional assumptions are required to guarantee that all the points are generated in the implicit mesh.

Assumption 3 *The set $\mathcal{D} = D$ of positive spanning sets is finite and the elements of D are of the form $G\bar{z}_j$, $j = 1, \ldots, |D|$, where $G \in \mathbb{R}^{n\times n}$ is a non-singular matrix and each $\bar{z}_j$ is a vector in $\mathbb{Z}^n$.*

Assumption 4 *Let D represent a finite set of positive spanning sets satisfying Assumption 3. The set $\mathcal{D}$ is so that the elements $d_k \in D_k \in \mathcal{D}$ satisfy the following conditions:*

1. *d_k is a non-negative integer combination of the columns of D.*

2. *The distance between x_k and the point $x_k + \alpha_k d_k$ tends to zero if and only if α_k does:*

$$\lim_{k \in K} \alpha_k \|d_k\| = 0 \iff \lim_{k \in K} \alpha_k = 0,$$

for any infinite subsequence K.

3. *The limits of all convergent subsequences of $\bar{D}_k = \{d_k/\|d_k\| : \ d_k \in D_k\}$ are positive spanning sets for $\mathbb{R}^n$.*

In addition to these, some assumptions regarding the step-size update are also needed.

Assumption 5 *Let $\tau > 1$ be a rational number and $m^{max} \geq 0$ and $m^{min} \leq -1$ integers. If the iteration is successful, then the step-size parameter is maintained or increased by considering $\alpha_{new} = \tau^{m^+}\alpha$, with $m^+ \in \{0, \ldots, m^{max}\}$. If the iteration is unsuccessful, then the step-size parameter is decreased by setting $\alpha_{new} = \tau^{m^-}\alpha$, with $m^- \in \{m^{min}, \ldots, -1\}$.*

Assumption 6 *At iteration k, the search step only evaluates points in*

$$M_k = \bigcup_{x \in E_k} \{x + \alpha_k D z : z \in \mathbb{N}_0^{|D|}\},$$

where E_k represents the set of all the points evaluated by the algorithm previously to iteration k.

We now arrive to our first result, originally established by Torczon [47] in the context of pattern search, and generalized by Audet and Dennis to GPS [5] and MADS [6].

Theorem 1 *Let Assumption 1 hold. Under one of the Assumptions 3 or 4, combined with Assumptions 5, 6 and $\rho(\cdot) \equiv 0$, GLODS generates a sequence of iterates satisfying*

$$\liminf_{k \to +\infty} \alpha_k = 0.$$

If, instead of using integer lattices, an approach based on requiring a sufficient decrease of the objective function value when adding points to the list is considered, there is more flexibility on the type of poll directions considered, on the search step, and on the strategy to update the step-size parameter. The following assumption is still required:

Assumption 7 *The distance between x_k and the point $x_k + \alpha_k d_k$ tends to zero if and only if α_k does:*

$$\lim_{k \in K} \alpha_k \|d_k\| = 0 \iff \lim_{k \in K} \alpha_k = 0,$$

for all $d_k \in D_k$ and for any infinite subsequence K.

The following result, first established by Kolda, Lewis and Torczon in GSS [26], can be derived:

Theorem 2 *Let Assumptions 1 and 2 hold. When ρ is a forcing function and Assumption 7 holds, GLODS generates a sequence of iterates satisfying*

$$\liminf_{k \to +\infty} \alpha_k = 0.$$

Thus, we conclude the first step in analyzing the convergence of GLODS. The next step is to ensure that there is a convergent subsequence of iterates and that the corresponding limit satisfies some stationarity result. In order to do this, we will focus on analyzing the behavior of GLODS in *refining subsequences*, a particular type of subsequences of unsuccessful iterations, first defined in GPS [5].

Definition 1 *A subsequence $\{x_k\}_{k \in K}$ of iterates corresponding to unsuccessful poll steps is said to be a refining subsequence if $\{\alpha_k\}_{k \in K}$ converges to zero.*

Convergent refining subsequences are a consequence of Theorems 1 and 2 and Assumption 1. Thus, when the conditions required to fulfill either Theorem 1 or 2 are met, GLODS will generate at least one convergent refining subsequence. The behavior of GLODS will be analyzed in its limit point along *refining directions*, another concept introduced in [6].

Definition 2 *Let x_* be the limit point of a convergent refining subsequence $\{x_k\}_{k \in K}$. If the limit $\lim_{k \in K'} d_k / \|d_k\|$ exists, where $K' \subseteq K$, $d_k \in D_k$, and $x_k + \alpha_k d_k \in \Omega$, for sufficiently large $k \in K'$, then this limit is said to be a refining direction for x_*.*

Since GLODS is intended for the minimization of non-smooth functions, a possible stationarity result would consist in establishing the non-negativity of the Clarke–Jahn generalized directional derivatives [23], computed for a limit point of the sequence of iterates generated by the algorithm, for the whole set of directions belonging to the Clarke generalized tangent cone to the feasible region [11]. To do this, the following definitions are required.

Definition 3 *A vector $d \in \mathbb{R}^n$ is said to be a Clarke tangent vector to the set $\Omega \subset \mathbb{R}^n$ at the point x in the closure of Ω if for every sequence $\{y_k\}$ of elements of Ω that converges to x and for every sequence of positive real numbers $\{t_k\}$ converging to zero, there exists a sequence of vectors $\{w_k\}$ converging to d such that $y_k + t_k w_k \in \Omega$.*

Definition 4 *The set $T_\Omega^{Cl}(x)$ of all Clarke tangent vectors to Ω at x is called the Clarke tangent cone to Ω at x.*

Definition 5 *A vector $d \in \mathbb{R}^n$ is said to be a hypertangent vector to the set $\Omega \subset \mathbb{R}^n$ at the point x in Ω if there exists a scalar $\epsilon > 0$ such that*

$$y + tw \in \Omega, \quad \forall y \in \Omega \cap B(x;\epsilon), \quad w \in B(d;\epsilon), \quad \textit{and} \quad 0 < t < \epsilon.$$

Definition 6 *The set $T_\Omega^H(x)$ of all hypertangent vectors to Ω at x is called the hypertangent cone to Ω at x.*

The Clarke tangent cone is the closure of the hypertangent cone, and the hypertangent cone is the interior of the Clarke tangent cone. The Clarke tangent cone is a generalization of the tangent cone commonly used in Nonlinear Programming (as in [36], Definition 12.2 and Figure 12.8).

Having defined the sets $T_\Omega^{Cl}(x)$ and $T_\Omega^H(x)$, we can now define the Clarke–Jahn generalized directional derivative.

Definition 7 *Let f be Lipschitz-continuous near a point $x \in \Omega$. The Clarke–Jahn generalized directional derivative, computed for f at x, for $d \in T_\Omega^H(x)$ is defined as:*

$$f^\circ(x;d) = \limsup_{\substack{x' \to x,\, x' \in \Omega \\ t \downarrow 0,\, x' + td \in \Omega}} \frac{f(x'+td) - f(x')}{t},$$

This definition can be extended to directions v belonging to the Clark tangent cone to Ω at x (Proposition 3.9 in [6]).

Definition 8 *Let f be Lipschitz-continuous near a point $x \in \Omega$. The Clarke–Jahn generalized directional derivative, computed for f at x, for $v \in T_\Omega^{Cl}(x) \setminus T_\Omega^H(x)$ is defined as:*

$$f^\circ(x;v) = \lim_{d \in T_\Omega^H(x), d \to v} f^\circ(x;d).$$

We now have all the necessary elements to define Clarke-criticality.

Definition 9 *Let f be Lipschitz-continuous near a point $x_* \in \Omega$. The point x_* is a Clarke critical point of f in Ω if, for all directions $d \in T_\Omega^{Cl}(x_*)$, $f^\circ(x_*;d) \geq 0$.*

Additionally, if the objective function f is strictly differentiable at x_*, meaning that the Clarke generalized gradient [11] is a singleton, namely $\nabla f(x_*)$, then this definition can be restated using the gradient vector.

Definition 10 *Let f be strictly differentiable at a point $x_* \in \Omega$. The point x_* is a Clarke-KKT critical point of f in Ω if, for all directions $d \in T_\Omega^{Cl}(x_*)$, $\nabla f(x_*)^\top d \geq 0$.*

We can now establish the final results regarding the convergence of GLODS.

Theorem 3 *Consider a refining subsequence $\{x_k\}_{k \in K}$ generated by GLODS, converging to $x_* \in \Omega$. Let $d \in T_\Omega^H(x_*)$ be a refining direction for x_* and f be Lipschitz-continuous near x_*. Under these circumstances, $f^\circ(x_*;d) \geq 0$.*

In other words, we have established stationarity along refining directions belonging to the hypertangent cone $T_{\Omega}^{H}(x_*)$. To establish that x_* is a Clarke critical point, we will require the asymptotic density of these refining directions in $T_{\Omega}^{H}(x)$. The following theorem states the result.

Theorem 4 *Consider a refining subsequence $\{x_k\}_{k \in K}$ generated by GLODS, converging to $x_* \in \Omega$. Let $T_{\Omega}^{H}(x_*) \neq \emptyset$ and f be Lipschitz-continuous near x_*. Assume that the set of refining directions for x_* is dense in $T_{\Omega}^{H}(x)$. Under these circumstances, x_* is a Clarke critical point of f in Ω.*

If strict differentiability of f at x_* is assumed, instead of Lipschitz-continuity near x_*, then from Theorem 3 it follows that $\nabla f(x_*)^\top d \geq 0$, for all refining directions in $T_{\Omega}^{H}(x_*)$. Consequently, from Theorem 4, is follows that x_* is a Clarke-KKT critical point of f in Ω. Thus, we conclude our analysis of the convergence of GLODS.

3

Radial Basis Functions

This chapter explains what Radial Basis Functions (RBFs) are and why they are interesting, not only to this work but also to others in global derivative-free optimization. Some challenges that arise from the use of these surrogate functions and our proposed approach to address them will also be detailed. Lastly, a short review of acquisition functions will be presented.

3.1 Definition

A radial basis function is a function $g : \mathbb{R}^n \to \mathbb{R}$ whose value depends only on the norm of the vector given as argument. In other words, g is a radial basis function if and only if:

$$\|x\| = \|y\| \implies g(x) = g(y), \quad \forall x, y \in \mathbb{R}^n \tag{3.1}$$

This property allows regarding function g as a composition of a function $h : \mathbb{R}_0^+ \to \mathbb{R}$ with a norm, meaning that $g(x) = h(\|x\|)$, $\forall x \in \mathbb{R}^n$. Function h is usually referred to as the *kernel* function. In particular, this means that these functions are much less computationally demanding than the original functions we want to optimize. This is one of the reasons why radial basis functions have been widely used in the literature as surrogate functions. The other main reason is because they act as rough models of the objective function, which allows predicting how a given objective function f behaves away from points already evaluated. These models are computed, resorting to numerical interpolation.

In its most simple form, the interpolating function s is defined as a sum of identical radial basis functions, centered at points already evaluated. Let x_i, $i \in \{1, \ldots, m\}$, be the points whose objective function values are already known. Function s is defined as

$$s(x) = \sum_{i=1}^{m} \lambda_i \, g(x - x_i), \quad \forall x \in \mathbb{R}^n. \tag{3.2}$$

As s is an interpolating function, it must then conform to the following conditions:

$$s(x_i) = f(x_i), \quad \forall i \in \{1, \ldots, m\}. \tag{3.3}$$

Combining (3.2) and (3.3) leads to the linear system of equations (3.4), where λ_i are the model coefficients to be determined:

$$\begin{bmatrix} g(x_1-x_1) & g(x_1-x_2) & \dots & g(x_1-x_m) \\ g(x_2-x_1) & g(x_2-x_2) & \dots & g(x_2-x_m) \\ \vdots & \vdots & & \vdots \\ g(x_m-x_1) & g(x_m-x_2) & \dots & g(x_m-x_m) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix} \tag{3.4}$$

For general functions g, solving this system would not be trivial or even possible. In fact, when handling linear combinations of general functions such as in (3.2), Theorem of Mairhuber–Curtis (see Theorem 1 in [30]) guarantees that there is at least one set of points $\{x_1,\dots,x_m\} \subset \mathbb{R}^n$, for $n \geq 2$, such that the matrix A of the form

$$A(\{x_1,\dots,x_m\}) := \begin{bmatrix} g(x_1-x_1) & g(x_1-x_2) & \dots & g(x_1-x_m) \\ g(x_2-x_1) & g(x_2-x_2) & \dots & g(x_2-x_m) \\ \vdots & \vdots & & \vdots \\ g(x_m-x_1) & g(x_m-x_2) & \dots & g(x_m-x_m) \end{bmatrix} \tag{3.5}$$

is singular. However, one of the key features of radial basis functions is that they allow the square matrix A to be *positive definite*. Positive definiteness implies that all eigenvalues of the matrix are strictly positive. Since the determinant of a matrix is the product of all its eigenvalues, and all eigenvalues of A are strictly positive, its determinant is also strictly positive. Therefore, A is invertible, guaranteeing that the system (3.4) is determined. Thus, the model coefficients can be uniquely determined. Positive definiteness partially results from defining g as a function of the norm; under these circumstances, A as in (3.5) is a distance matrix to which some continuous transformation was applied - the kernel function h. By being a distance matrix, it is guaranteed that it is non-singular, regardless of the set of points $\{x_1,\dots,x_m\} \subset \mathbb{R}^n$ considered. However, derivatives are not defined at these points. The transformation h solves this problem, granting it differentiability.

Depending on the kernel chosen, additional steps might be required in order to guarantee the non-singularity of A. A radial basis function g is said to be positive definite if all matrices A of the form (3.5) are positive definite, regardless of the set of points chosen. In this case, no further steps are needed. Otherwise, (3.2) should be augmented to incorporate a polynomial tail. In this case, g is said to be *conditionally* positive definite. The degree of the required polynomial depends on the family of kernels considered. Let $n \in \mathbb{N}$, $q \in \mathbb{N}$ and P^n_{q-1} be the space of all polynomials of n variables and of degree at most $q-1$. Let $p_1,\dots,p_q$ be a basis of this space.

The extension of the simple RBF model (3.2) by a polynomial tail is then defined as

$$s(x) = \sum_{i=1}^{m} \lambda_i \, g(x-x_i) + \sum_{j=1}^{q} \gamma_j \, p_j(x), \quad \forall x \in \mathbb{R}^n. \tag{3.6}$$

Additional constraints (3.7) are then considered, such that the system (3.8), resulting from the conjunction of (3.3), (3.6) and (3.7), is always perfectly determined:

$$\sum_{i=1}^{m} \lambda_i \, p_j(x_i) = 0, \quad \forall j \in \{1, \dots, q\}. \tag{3.7}$$

Orthogonality of λ to the basis $p_1, \dots, p_q$ is relevant when A is non-singular, despite g being only conditionally positive definite. In this case, the system (3.4) is perfectly determined. Thus, a polynomial tail is not required, as the goal of adding a polynomial tail is to fix the problem of system (3.4) not being determined (by fulfilling the conditional positive definiteness of g). As such, conditions (3.7) ensure that the polynomial tail is equal to zero in this scenario.

The augmented model and the orthogonality conditions can be combined into the following system:

$$\begin{cases} \sum_{i=1}^{m} \lambda_i \, g(x_k - x_i) + \sum_{j=1}^{q} \gamma_j \, p_j(x_k) = f(x_k), & \forall k \in \{1, \dots, m\}, \\ \sum_{i=1}^{m} \lambda_i \, p_j(x_i) = 0, & \forall j \in \{1, \dots, q\}. \end{cases} \tag{3.8}$$

This system can also be written in the matrix form

$$\begin{bmatrix} g(x_1 - x_1) & \dots & g(x_1 - x_m) & p_1(x_1) & \dots & p_q(x_1) \\ \vdots & & \vdots & \vdots & & \vdots \\ g(x_m - x_1) & \dots & g(x_m - x_m) & p_1(x_m) & \dots & p_q(x_m) \\ p_1(x_1) & \dots & p_1(x_m) & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ p_q(x_1) & \dots & p_q(x_m) & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \\ \gamma_1 \\ \vdots \\ \gamma_q \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.9}$$

which is more commonly written as

$$\begin{bmatrix} \mathbf{A} & \mathbf{P} \\ \mathbf{P}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \lambda \\ \gamma \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}, \tag{3.10}$$

where

$$\mathbf{P} := \begin{bmatrix} p_1(x_1) & \dots & p_q(x_1) \\ \vdots & & \vdots \\ p_1(x_m) & \dots & p_q(x_m) \end{bmatrix}, \ \lambda := \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix}, \ \gamma := \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_q \end{bmatrix}, \ \mathbf{f} := \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix}. \tag{3.11}$$

A conditionally positive definite radial basis function is said to be of *order* $q \in \mathbb{N}$ if $q - 1$ is the lowest degree of the polynomial tail that must be added to (3.2) so that (3.8) is uniquely solvable. When a radial basis function is positive definite, it is said to be of order 0, for ease of use. Naturally, different families of radial basis functions are of different orders (see Table 3.1).

3.1.1 An example

Let $x_1 = -4$, $x_2 = 1$, $x_3 = 3$, $f(x) = x(x-1)$, $g(x) = \|x\|^3$, $p_1(x) = x$, and $p_2(x) = 1$. The first step to calculate the interpolating function s is to build the system of linear equations as in (3.8), in order to determine the weights λ_i, for $i \in \{1,2,3\}$ and γ_j for $j \in \{1,2\}$.

$$\begin{bmatrix} g(x_1-x_1) & g(x_1-x_2) & g(x_1-x_3) & p_1(x_1) & p_2(x_1) \\ g(x_2-x_1) & g(x_2-x_2) & g(x_2-x_3) & p_1(x_2) & p_2(x_2) \\ g(x_3-x_1) & g(x_3-x_2) & g(x_3-x_3) & p_1(x_3) & p_2(x_3) \\ p_1(x_1) & p_1(x_2) & p_1(x_3) & 0 & 0 \\ p_2(x_1) & p_2(x_2) & p_2(x_3) & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ 0 \\ 0 \end{bmatrix}$$

Using the values given, it follows that

$$\begin{bmatrix} 0 & 125 & 343 & -4 & 1 \\ 125 & 0 & 8 & 1 & 1 \\ 343 & 8 & 0 & 3 & 1 \\ -4 & 1 & 3 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} 20 \\ 0 \\ 6 \\ 0 \\ 0 \end{bmatrix}.$$

Lastly, the system is solved and the values λ_i, for $i \in \{1,2,3\}$ and γ_j for $j \in \{1,2\}$ are determined:

$$\begin{cases} \lambda_1 = 0.05 \\ \lambda_2 = -0.175 \\ \lambda_3 = 0.125 \\ \gamma_1 = -1.25 \\ \gamma_2 = -6 \end{cases}$$

As such, the interpolating function s has the following expression:

$$s(x) = 0.05\|x+4\|^3 - 0.175\|x-1\|^3 + 0.125\|x-3\|^3 - 1.25x - 6$$

Figure 3.1 displays the plot of s, as well as each of its individual components, the known points $(x_i, f(x_i))$, $i \in \{1,2,3\}$, where (3.3) holds, and the objective function f.

3.2 RBF families

Radial basis functions can be grouped into families, sharing common properties. Table 3.1 displays some of the RBF families known, as well as their respective orders and parameters range.

In some expressions, there is a c parameter. This is usually regarded as the *shape* parameter, and dictates the *steepness* of the function: higher values of c result in stronger variations on the slope of the curves. Figure 3.2 displays a comparison of different values for the shape parameter c for a radial basis function of the Gaussian family. When c is

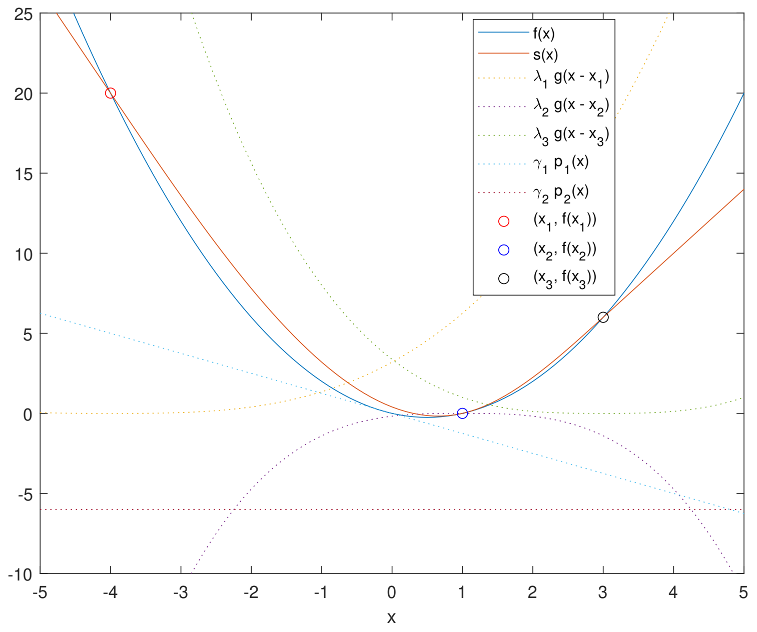

Figure 3.1: Plot of the interpolating function s, computed for function $f(x) = x(x-1)$, using the kernel $h(x) = x^3$.

Table 3.1: Some RBF families and their respective orders and parameters range.

Family	**Expression**	**Order**	**Parameters range**
Polyharmonic Splines I	$\|\|x\|\|^{2k-1}$	k	$k \in \mathbb{N}$
Polyharmonic Splines II	$\|\|x\|\|^{2k}\log(\|\|x\|\|)$	$k+1$	$k \in \mathbb{N}$
Gaussian	$e^{-c\|\|x\|\|^2}$	0	$c > 0$
Multiquadric	$(\|\|x\|\|^2 + c)^{\frac{k}{2}}$	$\left\lceil \frac{k}{2} \right\rceil$	$k \in 2\mathbb{N}-1,\ c > 0$
Inverse Multiquadric	$(\|\|x\|\|^2 + c)^{-\frac{k}{2}}$	0	$k \in \mathbb{N},\ c > 0$

high, the need to satisfy the interpolation conditions (3.3) leads to the *bed of nails* effect, which can be seen in Figure 3.3.

Naturally, both c and k parameters have a very noticeable impact on the quality of the interpolating function s. As such, care must be taken when choosing values for these parameters. The parameter k dictates the order of the radial basis function, and so, in order to guarantee the solvability of (3.8), higher degree polynomial tails are required. On the other hand, the parameter c has a direct impact on the accuracy of s when extrapolating, as well as on the conditioning of (3.8), by what is referred to as the *uncertainty principle* [42]: low values of c increase the accuracy of s, but in turn lead to ill-conditioned systems. Thus, c should be neither too high nor too low. Rippa [40] proposed an algorithm that dynamically calculates a good value for the shape parameter c, based on the set of points considered. Alternatively, this issue can be bypassed altogether by simply choosing one of the RBF families that do not include a shape parameter in their expressions, such as

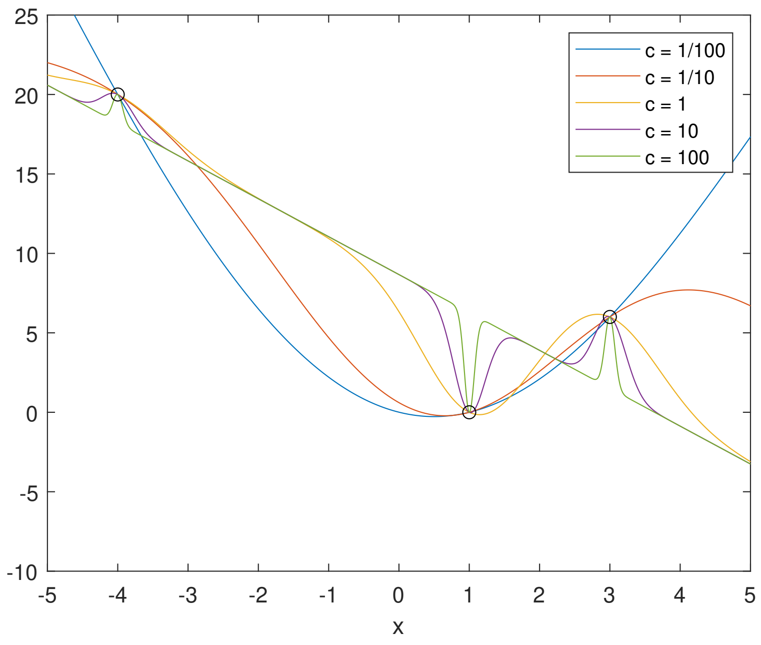

Figure 3.2: Comparison of different shape parameter values c for a radial basis function of type Gaussian.

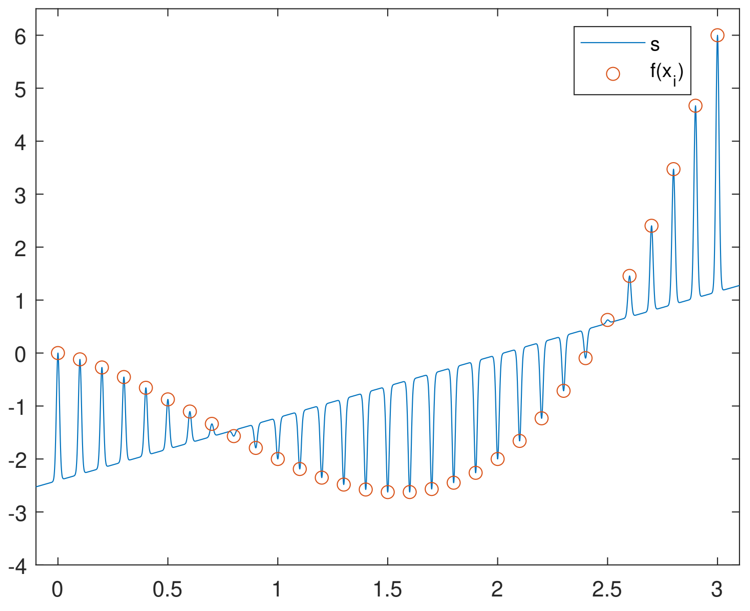

Figure 3.3: Showcase of the *bed of nails* effect in a radial basis function of the Gaussian family with a very high value of the shape parameter c, when applied to the function $f(x) = x^3 - 2x^2 - x$.

Polyharmonic Splines I and II.

3.3 Challenges when handling RBF models

Some issues arise when working with radial basis functions. One of such issues consists on which radial basis function should be considered. The answer to this question is not clear. *Cubic* radial basis functions - $g(x) = \|x\|^3$ - and *Thin-plate Splines* - $g(x) = \|x\|^2 \log(\|x\|)$ - are the most popular, and have been used with promising results in works such as ORBIT [50], MATSuMoTo [31], and GLIS [7]. Both are very attractive by their ease of use, as no shape parameter is required, and are of relatively low order, requiring only a linear polynomial tail to guarantee positive definiteness. However, the thin-plate spline expression does not admit second-order derivatives for $\|x\| = 0$. As such, the resulting interpolating function s will also not admit second-order derivatives for the same cases. Since we want to use derivative-based methods to minimize the RBF models, instead we considered the *Altered Thin-plate Splines* - $g(x) = \|x\|^4 \log(\|x\|)$, whose second-order derivatives can be extended by continuity to cover the case where $\|x\| = 0$.

The second issue we face is deciding what points should be chosen to compute the RBF models. Since GLODS performs most of the function evaluations at the poll step, it is expected that several points in the list are very close to each other. Due to their proximity, these points can be expected to carry similar information regarding the objective function. Moreover, using points too close to each other may also worsen the conditioning of the systems that need to be solved to compute the RBF models. As such, it is unwise to consider the entire list of points when building an interpolation model. Instead, a number of points should be selected based on their geometry and value to the interpolation model. Iske [22] suggests that a uniformly distributed set of points is best. In this work, we will consider three other strategies:

1. Choosing points in decreasing order of step-size (**ALPHA**);
2. Choosing active points first, in decreasing order of step-size, and then inactive ones, also in decreasing order of step-size (**ACTIVE**);
3. Choosing the origin and the best point of each local line of search, and, if more points are required, then selecting from the remaining points, also in decreasing order of step-size (**LINES**). Best points are chosen first, in decreasing order of step-size. Origin points are chosen after all best points have been selected, also in decreasing order of step-size;

All these strategies strive to choose points significantly distant from each other, scattered across the feasible region, while also providing the model with important information regarding the objective function. Strategy ALPHA chooses points in decreasing order of step-size because points with a large step-size are in either good or unexplored areas of the feasible region. Also, since their step-size is large, they are very likely to be far away from each other. Strategy ACTIVE provides a trade-off between the first strategy and giving the model the location of the best points found. Strategy LINES takes advantage of how GLODS' carries out local search by identifying each local line's origin and best points, and feeding these to the model. It is worth noting that when two lines are merged, the newest line is terminated. Thus, for the purpose of strategy LINES, GLODS considers the last active point of the terminated line to be one of the points to be selected when selecting each line's best point.

It is also important to consider how many points should be used for interpolation. On one hand, selecting too many points increases the chance of having them too close to each other. On the other hand, not selecting enough points most likely leads to insufficient information being fed to the model, resulting in very poor extrapolation accuracy. Therefore, it is important to select an adequate number of points. As per ORBIT [50], $n+2$ seems to be an appropriate lower bound to the number of points chosen, since the RBF models will include a linear tail. It may happen that this new algorithm (GLODS with the incorporation of RBFs in the search step, denoted by BoostGLODS from now on) attempts to perform a search step before attaining the imposed lower bound. In this situation, BoostGLODS will fall back to the original default search step of GLODS - Sobol sequences. Regarding the upper bound, a few different options will be considered:

1. $\frac{(n+1)(n+2)}{2}$ (**QUAD**);
2. $(n+1)(n+2)$ (**DOUBLEQUAD**);
3. $2(n+1)(n+2)$ (**QUADQUAD**);
4. $+\infty$, no upper bound is imposed (**INF**).

These are derived from the number of points required to build a complete quadratic interpolation model (expression QUAD). RBF models are more complex than quadratic models, and so it makes sense to also consider more points to build them. Despite the previous considerations, we decided to test not imposing an upper limit whatsoever (expression INF). However, our perception is that the INF variant will not yield good results.

Lastly, we require a method capable of minimizing the RBF models. Popular metaheuristics such as particle swarm, genetic algorithms, simulated annealing and tabu search [10] could be used. Since radial basis functions have well defined derivatives, we can also use derivative-based methods, which, from the theoretical point of view, are more robust and often more efficient than pure metaheuristics. We selected a few MatLab methods to incorporate and test in GLODS:

1. GlobalSearch (**GS**) [48];
2. MultiStart [1] using its default initialization (**rMS**) or using a user-defined initialization (**dMS**);

3. GlobalTR (**GTR**)[14].

GlobalSearch and MultiStart are both global derivative-based MatLab innate solvers, relying on the MatLab function `fmincon` for local constrained nonlinear optimization. This function corresponds to numerical implementations of a variety of algorithms. We will be using the SQP algorithm, described in Chapter 18 of [36]. MultiStart simply performs local search starting from a set of initial points, which by default is randomly generated. GlobalSearch is also random, and so using either of them would deprive GLODS of its deterministic nature. MultiStart, however, can be made deterministic by providing the method with a user-defined set of initial points. In other words, we can make MultiStart deterministic at the cost of having to decide which points should be used for its initialization. GlobalTR, on the other hand, is a deterministic algorithm, using an optimization approach based on trust-regions. The downside of GlobalTR is that it is much slower than the other alternatives. GlobalTR was designed to work for a generic objective function, spending a large portion of its running time in the estimation of the gradient and Hessian of the objective function, as seen in Figure 4.12 in [14]. However, in our particular case, the objective function we are interested in minimizing is a RBF. As such, we can calculate the expressions defining the gradient and Hessian matrix, and use these to replace the derivative-estimation methods in GlobalTR. This makes the algorithm much faster, albeit still slower than the other two approaches considered.

Regarding MultiStart's user-defined initialization, we chose to take advantage of the geometry induced by the point-choice strategies. Since the set of points selected to build each RBF model is expected to be sufficiently spread across the feasible region, these points should be good candidates from where to start the minimization of the RBF model. We also chose to keep the original search step philosophy of generating at most n points. As such, we provide MultiStart with the first n points from the set of points chosen to build the RBF model. Since MultiStart simply carries out local search initializing from these points, we can expect at most n local minima will be identified.

3.4 Acquisition functions

We would also like to test the use of acquisition functions in GLODS. In this section, we will detail the procedure adopted, largely based on the approach adopted in GLIS [7].

Acquisitions functions are functions that take on an already existing surrogate model and elaborate on it, promoting the exploration of promising areas in terms of function value or areas far away from already evaluated points. This provides a trade-off between using the information available and getting additional information by evaluating points in unexplored areas. Naturally, this does not mean acquisition functions are strictly superior to pure surrogate models; sometimes opting for a greedy strategy is a better approach than opting for a very thorough exploration of the feasible region, specially when there are budget constraints.

The first step in defining an acquisition function is to define inverse distance weighting functions. These will gauge how close a point x is to a point already in the list; the closer they are, the higher the value of this inverse function. The inverse distance weighting functions are defined for every evaluated point x_i, $i \in 1,\dots,m$, as:

$$w_i(x) = \frac{\psi(\|x - x_i\|^2)}{\|x - x_i\|^2}, \quad \forall x \in \mathbb{R}^n. \tag{3.12}$$

In this work, the expressions considered for ψ were $\psi(\alpha) = 1$ and $\psi(\alpha) = e^{-\alpha}$, $\alpha \geq 0$. The next step is to normalize w_i, scaling it to $[0,1]$, which leads to the following functions v_i:

$$v_i(x) = \begin{cases} 1, & x = x_i \\ 0, & x \neq x_i,\ x \in \{x_1,\dots,x_m\} \\ \frac{w_i(x)}{\sum_{j=1}^{m} w_j(x)}, & x \notin \{x_1,\dots,x_m\} \end{cases} \tag{3.13}$$

Functions v_i attempt to translate the proximity of x to the known points, on a normalized scale (between 0 and 1). If x has already been evaluated, then its matching function v_i will be equal to 1 and all other v_j functions, $j \neq i$, will be equal to 0. On the other hand, if x has not been evaluated yet, then v_i will be higher for points x is closest to, and lower for points further away. This is useful because it allows us to define a function u that measures the *uncertainty* of the base surrogate model s at point x:

$$u(x) = \sqrt{\sum_{i=1}^{m} v_i(x)\big(f(x_i) - s(x)\big)^2} \quad \forall x \in \mathbb{R}^n. \tag{3.14}$$

In other words, u measures the difference between the objective function value of the known points and the value extrapolated from the surrogate model at x, weighted by the proximity of x to the already evaluated points. More crudely, u indicates the level of trust that can be safely given to the surrogate model; higher values of u indicate higher levels of uncertainty in the model, and so less trust should be placed in it. Naturally, $u(x_i) = 0$, $\forall i \in \{1,\dots,m\}$.

In addition to the uncertainty function u, a function z that measures if x is in an unexplored area of the feasible region is required. We will again make use of the w_i functions defined earlier, as they carry the proximity information: if all w_i functions are of very low value, then x is far away from all known points. Thus, x is in an unexplored area of the feasible region. As such, the distance function z is defined as:

$$z(x) = \begin{cases} 0, & x \in \{x_1,\dots,x_m\} \\ \frac{2}{\pi}\arctan\left(\frac{1}{\sum_{i=1}^{m} w_i(x)}\right), & x \notin \{x_1,\dots,x_m\} \end{cases} \quad \forall x \in \mathbb{R}^n. \tag{3.15}$$

This provides us with a normalized scale for how far away x is from any already evaluated point. Since this scale is normalized, it needs to be properly scaled so that it is comparable with the values observed in s. As such, let

$$\Delta F = \max\left\{ \max_{i \in \{1,\dots,m\}} f(x_i) - \min_{i \in \{1,\dots,m\}} f(x_i),\ \epsilon \right\}, \tag{3.16}$$

where $\epsilon > 0$.

The acquisition function for minimization is finally defined as:

$$a(x) = s(x) - \alpha u(x) - \delta \Delta F z(x), \quad \forall x \in \mathbb{R}^n, \ \alpha, \delta \in \mathbb{R}_0^+, \ \epsilon \in \mathbb{R}^+, \tag{3.17}$$

where α and δ are parameters that aid in controlling how much weight is placed onto the non-surrogate model components of the acquisition function, and ϵ ensures that z does not vanish when the values $f(x_i)$ happen to be very similar. The component $\alpha u(x)$ promotes exploration of areas of the feasible region where there is uncertainty in the surrogate model, and $\delta \Delta F z(x)$ of areas that have not yet been explored.

As the acquisition functions in this work build upon a surrogate model based on RBFs, they share the same considerations detailed in Section 3.3 regarding which RBF family to choose, which and how many points to consider to build the RBF models, and how to minimize them. It is important to note that the s component is constructed using the points selected for interpolation, whereas the remaining components of the acquisition function use all of the points in the list. In fact, as explained in Section 3.3, using all the points in the list to compute s could be harmful to the quality of the model, since the points are likely to be very close to each other. On the other hand, functions u and z work on the proximity to known points. As such, by not considering the entire list of points for these components, we might be promoting the exploration of areas that actually have already been explored, simply because the acquisition model a did not know these areas had already been explored. Moreover, once the base surrogate model s is built, considering more points for the remaining components of the acquisition function does not pose additional difficulties. These other components do not suffer a significant loss of quality and efficiency from having more points to build them from, since there is no linear system to be solved for u and z.

3.4.1 An example

Figure 3.4 shows an acquisition function model, when applied to the example described in 3.1.1, now considering $\alpha = \delta = \epsilon = \frac{1}{4}$ and $\psi(\|x - x_i\|^2) = 1$. In this example, we can see that the acquisition function a looks much less like the original objective function than the surrogate model s. This is a deliberate decision and stems from trying to promote the exploration of other areas of the feasible region. In this particular example, the global minima for a is located near $x = -\frac{1}{2}$. This area is close to the best point evaluated so far, x_2, yet both values $u(-\frac{1}{2})$ and $z(-\frac{1}{2})$ are high. Value $z(-\frac{1}{2})$ is high because $x = -\frac{1}{2}$ is relatively far away from all points already evaluated. Value $u(-\frac{1}{2})$ is high because the values observed in the vicinity of $x = -\frac{1}{2}$ vary by a large amount, specially in the case of x_1. In fact, even though x_2 is much closer ($v_1(-\frac{1}{2}) = 0.1343$ and $v_2(-\frac{1}{2}) = 0.7313$), this is not enough to offset the difference in values observed, resulting in $u(-\frac{1}{2})$ being a high value. This is a symptom of not having enough points evaluated between x_1 and x_2. This does not happen as severely in the case of the local minima $x = 2$, as x_1 is much further away than it is from $x = -\frac{1}{2}$. The points in the vicinity of $x = 2$ are both much closer to it and their values in the objective function are also closer, resulting in much lower values for $z(2)$ and $u(2)$, respectively.

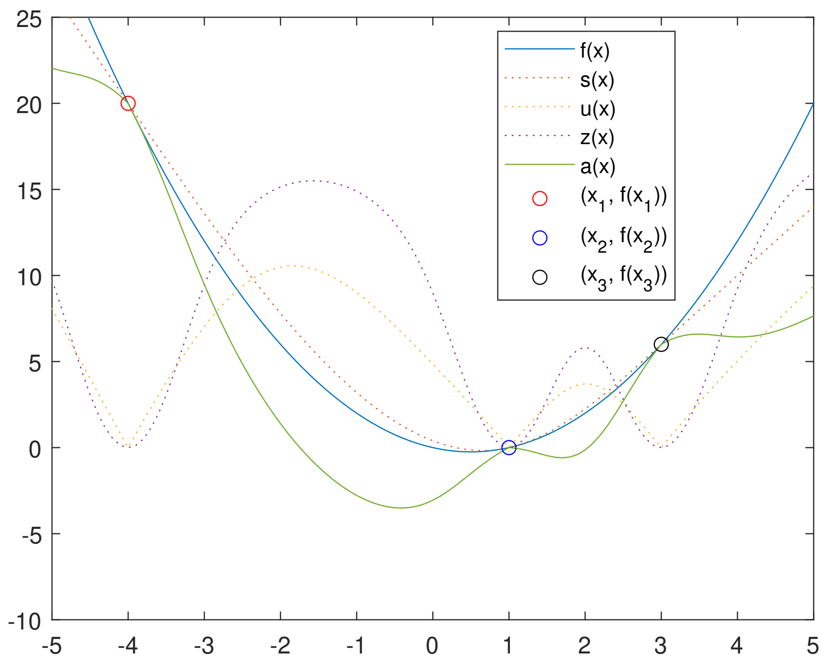

Figure 3.4: Plot of the acquisition function a computed for function $f(x) = x(x-1)$, considering $\alpha = \delta = \epsilon = \frac{1}{4}$ and $\psi(\|x - x_i\|^2) = 1$.

4 BoostGLODS

As previously mentioned, GLODS' search step is based on pseudo-random sampling techniques. As such, choosing where to start new local lines of search is largely left to chance. Using RBFs to model the objective function on a global scale should provide better quality to the search step. With this in mind, in this chapter we first detail BoostGLODS' search step. Then, we will revisit some of the decisions taken regarding when and how to use GLODS' search step, as well as introducing other new strategies to handle RBF models in BoostGLODS. It is important to note that changing the search step does not affect the convergence results of BoostGLODS.

4.1 BoostGLODS' search step

As previously stated, the main goal of the new search step is to make an informed decision of where to place new starting points in the feasible region, to initialize local lines of search. Our approach is based on RBFs, in an attempt to capture the global behavior of the objective function, based on points already evaluated in previous iterations. If the RBF models are accurate, then their local minima are good candidates to start new local lines of search. On the other hand, the models themselves are cheap to minimize, as detailed in Section 3.1. As such, BoostGLODS should remain an efficient algorithm.

The new search step can be described in a few steps:

1. Using one of the strategies ALPHA, ACTIVE or LINES, described in Section 3.3, and after deciding the maximum number of points to consider (QUAD, DOUBLEQUAD, QUADQUAD or INF), select a subset of points in the list;

2. Select a family of RBFs, and set values for the parameters k and c, if applicable. See Table 3.1 for the most widely used RBF families in the literature. In this work, we will be testing the RBFs $g(x) = \|x\|^3$ (**CUBIC**) and $g(x) = \|x\|^4 \log(\|x\|)$ (**aTPS**);

3. Select a polynomial tail for the RBF model, according to the parameter k of the family chosen. In this work, we will be testing a linear tail, where $p_1(x) = 1$ and

$p_2(x) = x$, since this is the usual procedure adopted in other works for the RBF families considered (see [31, 50]);

4. Compute the interpolating function s defined in (3.6), by solving the linear system of equations (3.8);

5. Minimize function s using one of the solvers presented in Section 3.3 (GS, rMS, dMS and GTR);

6. Call Algorithm 1 to possibly add the minima of function s to the list of points.

It is worth noting that, for aTPS, in order to guarantee the solvability of the linear systems, which allows computation of the model coefficients, a quadratic tail is required, according to Table 3.1. However, incorporating a quadratic tail would require additional considerations regarding the number of points chosen, as $n+2$ points are no longer enough. As such, we will postpone the use of a quadratic tail in aTPS until other tests have been conducted.

As a failsafe, when enough consecutive search steps result in no new points added to the list, disregarding any poll steps performed between search steps, BoostGLODS will attempt to generate new points using Sobol sequences, whenever the search step launching criteria are met. When a search step based on Sobol sequences is successful, BoostGLODS will return to RBFs in its search step.

In the next sections, we will discuss new ideas on how to incorporate this new search step into BoostGLODS.

4.2 Search step launching criteria

The first decision is related to the frequency of the search step. In the original GLODS, a search step is performed whenever there is only one active point in the list. This translates to roughly 5% of the iterations of a given GLODS' run including a search step, usually concentrated more towards the end. Increasing the number of search steps performed and/or spreading them more evenly across a run of the algorithm could yield better results. As such, additional criteria are proposed in BoostGLODS to perform a search step:

1. The number of active points in the list becomes equal to or falls below a threshold set by the user (**MINACT** followed by the threshold);

2. A number of consecutive iterations have been unsuccessful (**CONSEC** followed by the threshold);

3. A number of iterations - not necessarily consecutive - have been unsuccessful (**JUSTSUC**, again followed by the threshold).

For all these strategies, the threshold acts as a tuning knob to set how early in the run the search step is performed. For example, MINACT4 will start performing search steps much earlier than MINACT2. Similarly, CONSEC3 will perform fewer search steps than CONSEC2. Strategies MINACT and CONSEC were already implemented in the original GLODS. Strategy JUSTSUC is exclusive to BoostGLODS. Strategy MINACT reactively performs the search step when the local lines of search start stagnating. This is a very conservative strategy that only initializes new local lines of search when the number of active points in the list runs low. As such, a large portion of the search steps performed will be concentrated in the last iterations of the run. Strategy CONSEC proactively performs search steps when lines show some level of failure in a row. In general, strategy CONSEC starts performing search steps much earlier in the run than MINACT, but might not necessarily aggressively try to start new local lines of search as the existing ones stagnate. Instead, CONSEC tries to prevent the number of active points in the list from running low. Finally, strategy JUSTSUC is a new strategy designed to start performing search steps even earlier than CONSEC, as the criteria is much easier to meet.

4.3 Algorithmic flow strategies and RBF minima handling

Another important aspect of RBFs in BoostGLODS that will be explored in this work consists in the use of the RBF minima, generated in each search step. If these minima are placed in good regions of the feasible region - which relates to the accuracy of the RBF models -, it makes sense to explore them before the points GLODS typically selects as poll centers. Moreover, it would also make sense to switch the main drive of the algorithm from the poll step to this new and improved search step. To answer these questions, in this section we will discuss four different strategies, divided into two main categories, search-based strategies and poll-based strategies, depending on which step is meant to be the main drive of the algorithm.

For search-based strategies, we will test one strategy entitled **SEARCH**, which performs a search step at every iteration, and a poll step only when the search step is unsuccessful. Since the search step is performed at every iteration, the search step launching criteria discussed in Section 4.2 are irrelevant. Strategy SEARCH is closer to how RBF models are typically handled by optimization algorithms. A model is built, minimized, and refined using its old minima over and over again, until it is no longer successful.

For poll-based strategies, a few more options are available, and they can be combined independently with the strategies defined in Section 4.2. These strategies will differ mostly in how points generated at the search step are treated when selecting a point as a poll center. Three options have been considered:

1. The active point, not yet identified as a local minima, with the lowest objective function value is chosen as the poll center (**NOPRIO**). This is the default strategy in the original GLODS;

2. The active points generated at the search step are preferred when choosing a poll center (**POINTPRIO**). When no such point exists, BoostGLODS falls back to strategy NOPRIO until a new successful search step is performed. When more than one active point is added to the list in a search step, BoostGLODS will poll around them in increasing order of objective function value;

3. The best two points generated at the search step, as well as their successful offspring, are preferred when choosing a poll center (**LINEPRIO**). As with POINTPRIO, when no such points exist, BoostGLODS falls back to strategy NOPRIO until a new successful search step is performed.

Strategies POINTPRIO and LINEPRIO are an attempt to give the RBF minima priority over the standard NOPRIO strategy, recognizing that these points might be located in more interesting areas of the feasible region. This would allow BoostGLODS to reach the global minimum faster than GLODS, or to discover additional local minima. Strategy POINTPRIO gives priority just to the points resulting directly from the search step. In other words, polling around the RBF minima is done quickly, and the algorithm hastily resumes the NOPRIO strategy once all RBF minima have been selected as a poll center at least once. By doing this, it can happen that one of the RBF minima offspring becomes the best point in the list. In this case, even if the algorithm is back to selecting poll centers only based on objective function values, it can still choose the RBF minima offspring as poll centers (if these are better than the points previously in the list). Strategy LINEPRIO forces this behavior, by imposing polling around the best RBF minima and their offspring, until the poll step is unsuccessful. This makes BoostGLODS explore the same local line of search (whose starting point was a RBF minima) until it generates an unsuccessful poll step, even if the points evaluated are of worse value than that of the ones previously in the list. This procedure is only applied to the best two RBF minima because of the interaction with the strategies considered for search step launching. In the case of CONSEC2 strategies, at least the two best RBF minima and their successful offspring are explored, before another RBF model is built.

5

Performance Assessment Tools

In this chapter, we will detail the tools used to measure the performance of Boost-GLODS, as well as of other state-of-the-art solvers used to benchmark the proposed algorithm. We will begin by describing the collection of problems tested and some considerations pertaining to it. Then, we will briefly explain the tools widely used in the literature to analyze the performance of algorithms. Finally, we will propose a new tool that aims to clarify small differences in performance when the previous are not enough.

5.1 Test set

The first tool required to assess the performance of any solver is a problem collection. Table 5.1 contains all the problems considered in the computational experiments, as well as their dimensions (n), lower (l) and upper (u) bounds, and the number of local (loc) and global ($glob$) minima known. When only one number is stated in Table 5.1 for a bound, then it is assumed that this number is the common bound for all individual x_i, $i \in \{1,\dots,n\}$. For example, in the *ackley* problem, $l = -30$ and $u = 30$, meaning that

$$-30 \leq x_i \leq 30, \;\; \forall i \in \{1,\dots,n\}.$$

On the other hand, for example in the *branin_hoo* problem, $l = [-5,0]^\top$ and $u = [10,15]^\top$, meaning that

$$\begin{aligned} -5 \leq x_1 &\leq 10, \\ 0 \leq x_2 &\leq 15. \end{aligned}$$

Table 5.1: Collection of problems used for testing.

Problems	***n***	***l***	***u***	***loc***	***glob***
ackley [3]	10	-30	30	–	1
aluffi_pentini [3]	2	-10	10	2	1
becker_lago [3]	2	-10	10	4	4

bohachevsky [3]	2	-50	50	–	1
branin_hoo [21, 37]	2	$[-5\ 0]^\top$	$[10\ 15]^\top$	3	3
cauchy [8]	4	3	17	–	–
cauchy [8]	10	2	26	–	–
cosine_mixture [8]	2	-1	1	–	–
cosine_mixture [8]	4	-1	1	–	–
dekkers_aarts [3]	2	-20	20	3	2
epistatic_michalewicz [3]	5	0	π	–	1
epistatic_michalewicz [3]	10	0	π	–	1
exponencial [8]	2	-1	1	–	1
exponencial [8]	4	-1	1	–	1
fifteenn_local_minima [8]	2	-10	10	15^2	1
fifteenn_local_minima [8]	4	-10	10	15^4	1
fifteenn_local_minima [8]	6	-10	10	15^6	1
fifteenn_local_minima [8]	8	-10	10	15^8	1
fifteenn_local_minima [8]	10	-10	10	15^{10}	1
fletcher_powel [8]	3	-10	10	–	–
goldstein_price [8]	2	-2	2	4	1
griewank [20, 44]	5	-600	600	–	1
griewank [44]	10	-400	400	–	1
gulf [3]	3	$[0.1\ 0\ 0]^\top$	$[100\ 25.6\ 5]^\top$	–	1
hartman_4 [8]	3	0	1	4	1
hartman_4 [8]	6	0	1	4	1
hosaki [3]	2	$[0\ 0]^\top$	$[5\ 6]^\top$	2	1
kowalik [3]	4	0	0.42	–	1
langerman [3]	10	0	10	–	1
mccormick [3]	2	$[-1.5\ -3]^\top$	$[4\ 3]^\top$	2	1
meyer_roth [8]	3	-10	10	–	–
miele_cantrell [8]	4	-10	10	–	1
multi_gaussian [3]	2	-2	2	5	1
neumaier2 [3]	4	0	4	–	1
neumaier3 [3]	10	-100	100	–	1
odd_square[3]	20	-15	15	–	1
paviani [3]	10	2.001	9.999	–	1
periodic [3]	2	-10	10	50	1
poissonian [8]	2	$[1\ 1]^\top$	$[21\ 8]^\top$	–	–
powell [3]	4	-10	10	1	1
rastrigin [3]	10	-5.12	5.12	–	1
rosenbrock [8, 21, 44]	2	-5.12	5.12	1	1
rosenbrock [8, 21, 44]	6	-1000	1000	1	1

rosenbrock [8, 21, 44]	10	-2.048	2.048	1	1
salomon [3]	5	-100	100	–	1
salomon [3]	10	-100	100	–	1
schaffer1 [3]	2	-100	100	–	1
schaffer2 [3]	2	-100	100	–	1
schwefel [3]	10	-500	500	–	1
shekel_45 [8]	4	0	10	5	1
shekel_47 [8]	4	0	10	7	1
shekel_410 [8]	4	0	10	10	1
shekel_foxholes [3]	5	0	10	–	1
shekel_foxholes [3]	10	0	10	–	1
shubert [3]	2	-10	10	760	18
sinusoidal [3]	10	0	180	–	1
sixhumpcamel [8, 21]	2	$[-3\ -2]^\top$	$[3\ 2]^\top$	6	2
sphere [20, 44]	3	-5.12	5.12	1	1
storn tchebychev [3]	9	-128	128	–	1
tenn_local_minima [8]	2	-10	10	10^2	1
tenn_local_minima [8]	4	-10	10	10^4	1
tenn_local_minima [8]	6	-10	10	10^6	1
tenn_local_minima [8]	8	-10	10	10^8	1
three_hump_camel [3]	2	-5	5	3	1
transistor [3]	9	-10	10	–	1
wood [8]	4	-10	10	–	1

Figure 5.1 summarizes the dimensions in the collection of problems considered in this work. An important detail about these problems is that around 30% of them have their global minimum in the center point of the respective feasible region. Some algorithms include this point in their initialization, as is the case with GLODS and BoostGLODS (see Section 2.1.2.1). As such, using the collection of problems with this initialization option leads to bias in the results - the global minimum is reached in the initialization step for 30% of the problems considered.

For reasons that will later be detailed, we decided to divide the performance analysis into two different stages: Stage 1, where we only compare different versions of Boost-GLODS (see Chapter 6), and Stage 2, where we compare the best version of BoostGLODS with state-of-the-art solvers (see Chapter 7). In Stage 1, the initialization of BoostGLODS was changed to Sobol sequences. This way, we no longer forcibly include the center point in the initialization step, and the bias in the results is removed. In Stage 2, since we are using algorithms from other authors that have been extensively tested in their respective works, we would like to keep all of their default settings. As such, we decided to introduce some minor modifications in the collection of problems, instead of changing the algorithms' initialization steps. In the original GLODS article [15], the bounds of each

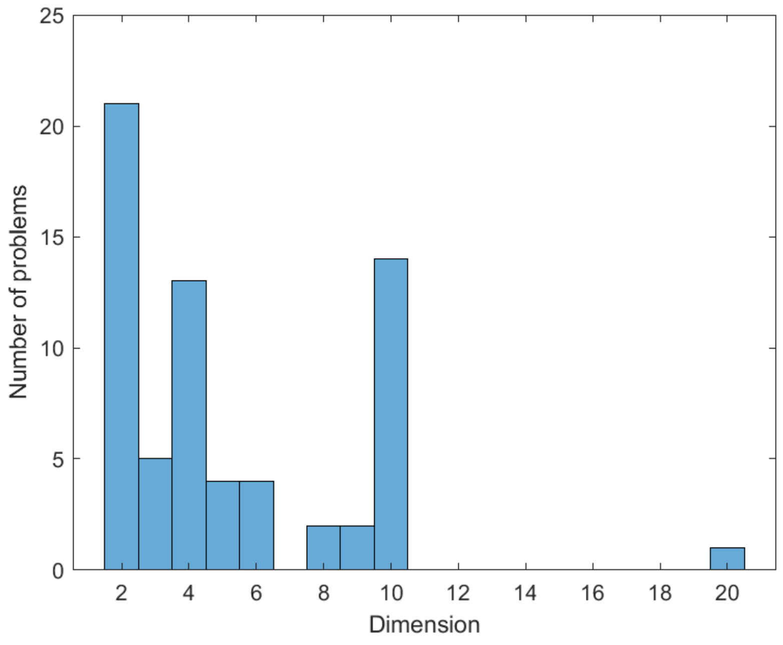

Figure 5.1: Distribution of the dimension of the problems in the test set.

problem were perturbed, by shrinking the upper bound by 35% of its distance to the lower bound. In Stage 2 of this work, we instead chose to move both the lower and upper bounds. This new collection is called the translated collection, and problems therein are of the form

$$\begin{array}{ll} \min & f(x_1,\ldots,x_n) \\ \text{s. t.} & l_i \leq x_i - \dfrac{(-1)^i}{3}\dfrac{u_i - l_i}{2} \leq u_i, \quad \forall i \in \{1,\ldots,n\}, \end{array}$$

where l_i and u_i are the lower and upper bounds of variable x_i, respectively.

It is important to note that, since we split the performance assessment of BoostGLODS into two different stages, each using its own collection of problems, Stage 2 provides a fairer ground of comparison in regards to other solvers, as it is not the collection of problems for which we selected the best version of BoostGLODS. Also, as we are moving the bounds outside its original definition, some feasible regions may now include points where the objective function is not defined. Namely, problems *gulf* and *paviani* were disregarded in Stage 2. Finally, since the default initialization of GLODS provides better results than Sobol sequences (see Chapter 7), we will once again initialize BoostGLODS with points in a line segment.

5.2 Data profiles

Data profiles [33] are a tool proposed by Moré and Wild to assess the performance of different solvers, specifically developed for derivative-free optimization. In this scientific domain, it is expected that the time required to evaluate the objective function largely out-weights the computational time of any other calculations. Thus, data profiles measure performance in terms of number of function evaluations required to solve a problem; the better the solver's performance, the less function evaluations it requires.

Let $\mathcal{P}$ be a set of problems and $\mathcal{S}$ be a set of solvers. A fundamental concept in defining data profiles is when to consider a problem $p \in \mathcal{P}$ successfully solved by solver $s \in \mathcal{S}$. For that, the decrease measured relatively to the initialization is compared to the decrease obtained by the best value known for the problem (f_l). In more detail, solver s solved problem p if

$$f(x^1) - f(x^{best}) \geq (1-\tau)\big(f(x^1) - f_l\big), \tag{5.1}$$

where x^1 is the first point evaluated by s, x^{best} is the best point found by s, and τ is the level of precision wanted. Higher levels of precision, corresponding to lower levels of τ, require larger decreases in order for a problem to be considered solved. In this work, unless otherwise stated, the precision level considered is $\tau = 10^{-5}$. Define $h_{p,s}$ as the minimum number of functions evaluations required by solver s to satisfy (5.1) for problem p. A data profile $d_s(\sigma)$ is defined by

$$d_s(\sigma) = \frac{1}{|\mathcal{P}|}\left|\left\{p \in \mathcal{P} : \frac{h_{p,s}}{n_p + 1} \leq \sigma\right\}\right|, \tag{5.2}$$

where n_p is the dimension of problem p. In other words, a data profile $d_s(\sigma)$ represents the percentage of problems in $\mathcal{P}$ that solver s solved, requiring at most a number of function evaluations equivalent to σ simplexes (sets of $n+1$ function evaluations).

Figure 5.2 displays an example of data profiles. In this case, within the budget $\sigma = 1000$, solver A solved around 60% of the problems in the collection, solver B around 70%, and solver C about 50%. Overall, solver B is clearly superior at every budget. Solver C eventually surpasses solver A at very large budget levels, albeit not by a very significant percentage.

It is worth noting that, as seen in this example, it is not always verified that $d_s(\sigma) \to 1$ as $\sigma \to \infty$. This means that not all problems are solvable by any of the solvers compared, regardless of the budget considered. This is specially the case in this work, as we decided to incorporate the information available on each problem's global minimum into all data profiles computed in Stage 1. The best value obtained for each problem, for any strategy tested, is kept and used as f_l, instead of limiting f_l to the best value among the versions currently being compared.

5.3 Performance profiles

Performance profiles [17] are another tool we will use to measure the performance of BoostGLODS. These were developed for general nonlinear optimization algorithms. Unlike data profiles, performance profiles do not limit themselves to measuring performance based on the number of function evaluations required. Other metrics, such as

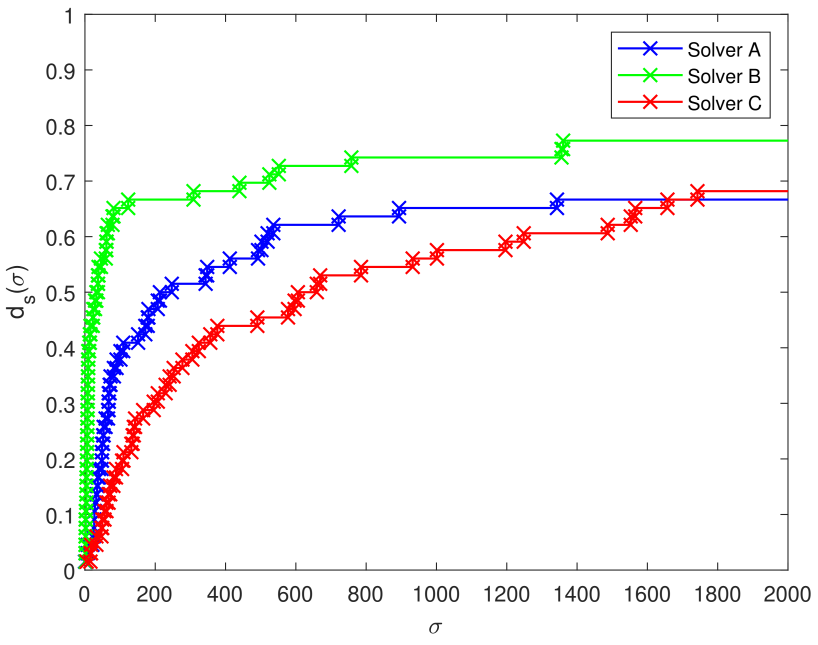

Figure 5.2: Example of data profiles.

execution time or number of local minima identified, can be considered. However, it is assumed that the metric considered only takes strictly positive values and that the lower the value of the metric, the better the performance is. This means that, for example, when considering as metric the number, m, of local minima identified, since a higher value corresponds to a better performance, the metric is adapted to $\frac{1}{m}$. This way, we heavily penalize low counts for the local minima found.

A performance profile is a function defined based on ratios. Let $t_{p,s}$ denote the value of the metric considered for solver $s \in \mathcal{S}$ and problem $p \in \mathcal{P}$. A ratio is computed between this value and the minimum value obtained for the same metric by any solver for problem p:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}. \tag{5.3}$$

The performance profile $\rho_s(\alpha)$ corresponds to the percentage of problems in $\mathcal{P}$ whose performance ratio $r_{p,s}$ is within a factor α of the best possible ratio:

$$\rho_s(\alpha) = = \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} : r_{p,s} \leq \alpha \right\} \right|, \quad \alpha \geq 1. \tag{5.4}$$

This definition of ratios and performance profiles leads to two important properties when comparing different solvers. The first is that

$$r_{p,s} = 1 \Leftrightarrow t_{p,s} = \min\{t_{p,s} : s \in \mathcal{S}\}. \tag{5.5}$$

In other words, the ratio for problem p and solver s is equal to one if and only if s presents the best performance out of all the solvers in $\mathcal{S}$, when solving problem p. This, in turn, leads to $\rho_s(1)$ representing the percentage of problems for which solver s is the best (the *efficiency* of solver s). The second property is shared with data profiles, where $\rho_s(\alpha)$ as $\alpha \to \infty$ indicates the percentage of problems that solver s is able to solve, for the stopping criteria considered. This is called the *robustness* of solver s.

Figure 5.3 displays an example of performance profiles, where solver C is the most efficient of the three solvers, and is as robust as solver B.

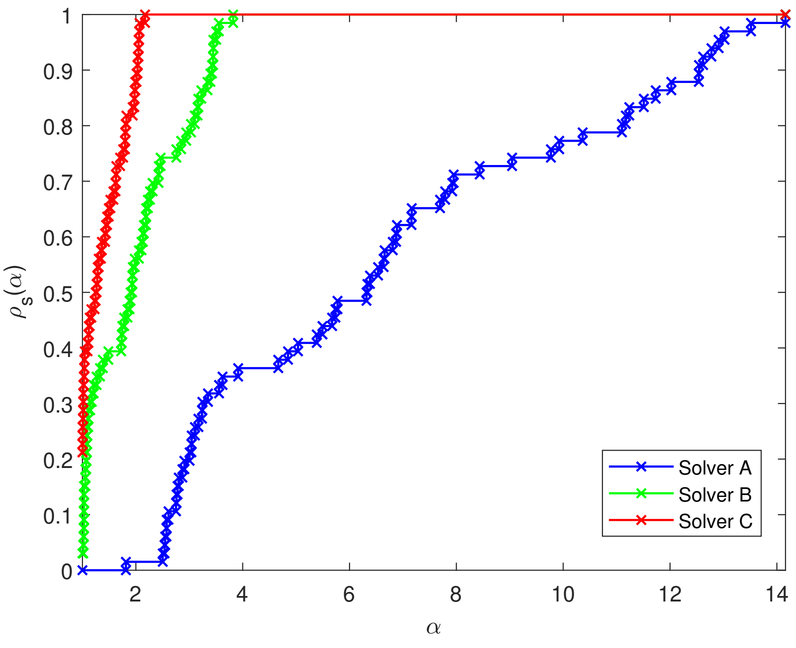

Figure 5.3: Example of performance profiles.

5.4 Scoring graphs

Often, when using data profiles to compare the performance of similar versions of a solver, the resulting graph would have several overlapping lines. This makes it very hard to decide which version is superior, for a given budget. Figure 5.4 displays an example of this. As such, an additional tool that allows us to decide which version is the best would be very helpful. With this purpose, we developed the *scoring graphs*.

A scoring graph takes a number of pre-built data profiles, gives each one a score η based on their performance for all budget levels, selects one as the *baseline solver* and then compares all other solvers' scores with the score of the baseline solver. The score for a solver s is given by

$$\eta_s(\omega) = \frac{1-\omega}{l_1}\int_0^{l_1} d_s(\sigma)d\sigma + \frac{\omega}{l_2 - l_1}\int_{l_1}^{l_2} d_s(\sigma)d\sigma, \tag{5.6}$$

where $0 \le l_1 < l_2$ are budget thresholds, and $0 \le \omega \le 1$ is a weight. In other words, the score $\eta_s(\omega)$ is a normalized and weighted sum of the areas under the data profile for

$\sigma \in [0, l_1]$ and $\sigma \in [l_1, l_2]$. Values for l_1 and l_2 should be set according to what is considered a low budget and the maximum budget. In this work, we set $l_1 = 750$, because this is where we see the most improvement in the data profiles for GLODS, and $l_2 = 2000$, because beyond $\sigma = 2000$, typically there are very few performance swings. In fact, in our tests, we impose a limit of 20000 function evaluations to any solver run. Since the computation of a data profile budget σ comes scaled by $\frac{1}{n_p+1}$, for example, for a problem with $n_p = 10$, then, for $\sigma \geq \frac{20000}{10+1} \approx 1818$, no more function evaluations are allowed. Therefore, no change is expected in the data profiles beyond this point; the algorithms have already stopped. If n_p is lower, then the number of function evaluations allowed is reached later. However, experience tells us that problems of lower dimensions are usually easier to solve. In fact, for most problems of lower dimension, GLODS meets the stopping criteria related to the step-size of active points before reaching the maximum of function evaluations. As such, the majority of these problems will be solved within the [0, 2000] interval. Considering the dimensions in Figure 5.1, we believe $l_2 = 2000$ is a good cutoff.

After calculating the score for each data profile, the scoring graph is then populated with the relative scores between each solver s and the baseline solver A, $\delta_s(\omega)$,

$$\delta_s(\omega) = \frac{\eta_s(\omega) - \eta_A(\omega)}{\eta_A(\omega)} \cdot 100. \tag{5.7}$$

These scores $\delta_s(\omega)$ represent the relative increase (or decrease) in performance between solver s and the baseline solver A at each weight ω. Large positive values indicate a substantial increase.

Figure 5.5 displays the scoring graph resulting from the data profiles in Figure 5.4, selecting solver B as the baseline solver. It is much easier to see the differences in performance in the scoring graph. In this example, solver C is the best at most weights ω, being only surpassed by solver E for very low budgets. This could also be seen in the data profile, as for $\sigma < 300$, E is better than C. Solver C remains the best for $\sigma > 300$, having a score slightly higher than A for all weights due to the slight advantage it has before reaching $\sigma = 500$. In this case, as we consider the weights where E is superior to C too low ($\omega < 0.2$), we would unanimously choose C as the best solver in this test.

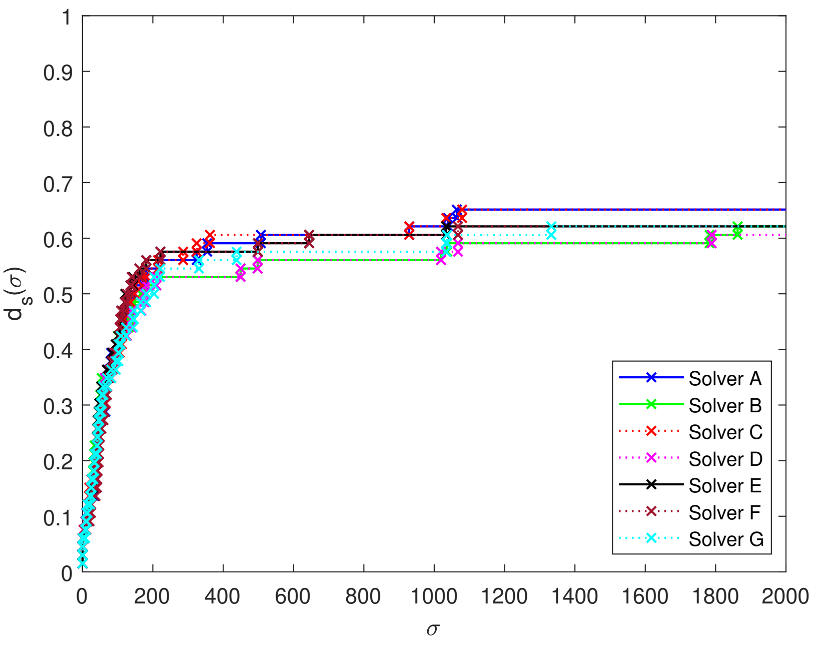

Figure 5.4: Example of overlapping data profiles.

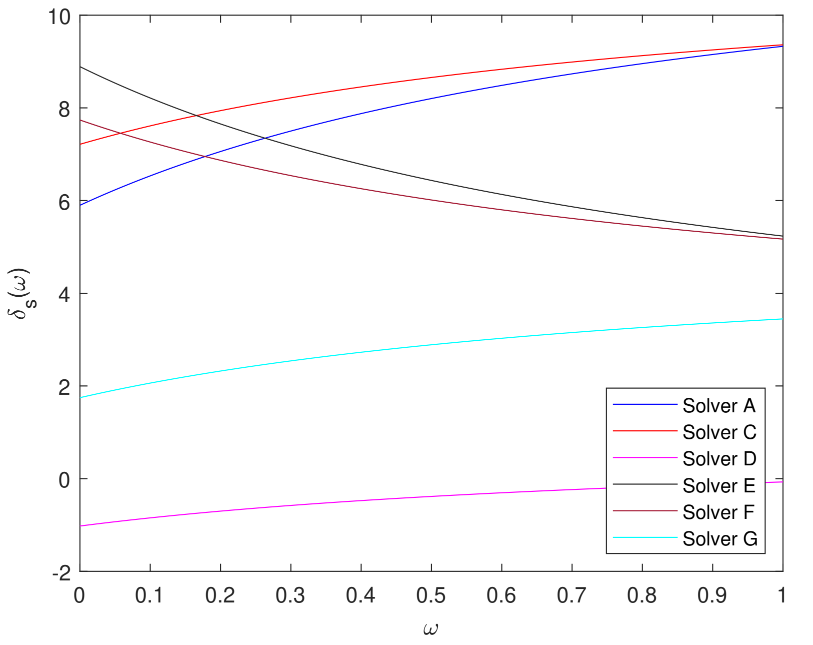

Figure 5.5: Example of a scoring graph.

6 Calibration of BoostGLODS

This chapter will detail the study that was conducted in order to narrow down the many possibilities for the final version of BoostGLODS, corresponding to the version with the best numerical performance.

We will begin by choosing an initial version as close as possible to the original GLODS. This version will use all GLODS default settings and simply replaces the use of Sobol sequences with RBF models in the search step. Then, we will venture into the many decisions we have to take regarding all the necessities to incorporate RBF models in BoostGLODS.

Since the number of versions available prevents us from testing all of them at once, we decided to conduct the study by making one decision at a time for each strategy, occasionally grouping more than one together if the number of possible combinations is low enough. As mentioned in Section 3.3, we will be using the CUBIC ($g(x) = \|x\|^3$) and aTPS ($g(x) = \|x\|^4 \log(\|x\|)$) RBF families. Since we cannot guarantee that both these types of models will perform similarly according to the strategies presented in Section 3.3 and Chapter 4, we will study them independently. When all options have been tested for both, we will finally compare each study's final version, and select the best. Finally, for the aTPS version, we will try replacing the linear tail in the model with a quadratic tail, more in line with the theoretical requirements of this RBF family. The assessment of the performance of acquisition models applied to the best version of BoostGLODS will also be considered.

As previously detailed in Section 2.1.2.3, the default settings of GLODS are to launch a new search step when there is only one active point in the list (MINACT1), to select the poll center corresponding to the active point with step-size above the stopping threshold and the lowest objective function value (NOPRIO), and to provide no special treatment to the points resulting from the search step. Also, GLODS initializes with points in a line segment, but as previously explained in Section 5.1, at this stage we decided to change this initialization to Sobol sequences so that the results were not influenced by the test set. Moreover, strategies to choose which points to build RBF models with were necessary. For that purpose, we considered ALPHA (just selects points in decreasing order of step-size), which is the simplest and easiest strategy to start with. Regarding the number of points to build the RBF models, we select $n+2$ as the lower limit and DOUBLEQUAD ($(n+1)(n+2)$) as the upper limit. For the RBF minimization method, there is no need to select one for the initial version, as this is the first setting we will be testing.

The characteristics of the initial version of BoostGLODS are summarized as follows:

- Initialization: Sobol sequences, generating $n+1$ points when n is even, and n otherwise;
- Search step launching criteria: MINACT1;
- Algorithmic flow strategy: NOPRIO;
- RBF models: CUBIC or aTPS (we will study both independently);
- RBF solver: (not applicable, since it will be the first setting tested);
- RBF point selection: ALPHA;
- RBF upper limit: DOUBLEQUAD.

6.1 Minimization method for RBFs

In Section 3.3, we proposed four different methods to minimize RBF models in BoostGLODS: GlobalSearch (GS), MultiStart with a deterministic initialization (dMS), MultiStart with a random initialization (rMS), and GlobalTR (GTR). The GS and rMS methods have random components in their algorithmic structure, whereas dMS and GTR are deterministic. Figure 6.1 shows the data profiles for the first batch of versions tested, where each version only differs from the initial version of BoostGLODS, detailed in the previous section, in the method used to minimize RBF models.

These data profiles indicate a similar performance for the variants proposed, with GS pulling ahead by a small margin. However, since GS is a random algorithm, we would like to make sure that GS does not present high variance before proceeding and selecting GS as the default solver to minimize RBF models. Figure 6.2 displays data profiles corresponding to ten runs of BoostGLODS using GS as the RBF solver. As can be seen, the results do not vary significantly. However, this can be explained by the search step launching criteria, as MINACT1 results in an average of 5% of the iterations performing a search step, which is a rather low percentage. In future tests, this percentage could increase, depending on the strategy selected to launch the search step. So, we can expect the variance induced by using GS to also increase. Of the deterministic solvers, GTR is slightly better in terms of percentage of problems solved than dMS. However, as indicated in Figure 6.3, GTR is clearly much slower, limiting the use of strategies that make a more intensive use of the search step. Using different search step launching criteria increases the number of search steps performed in a run, which in turn increases

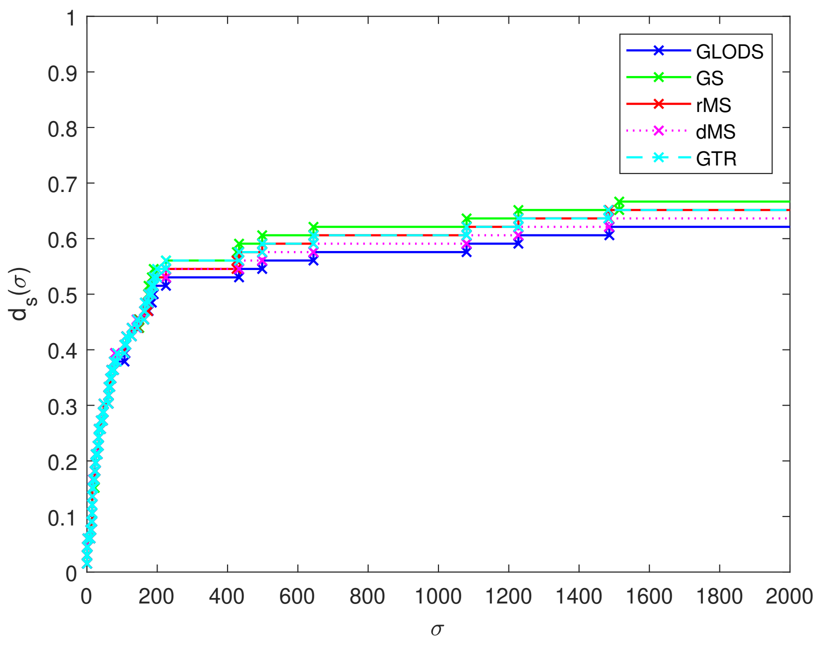

Figure 6.1: Data profiles corresponding to the use of different methods for RBF minimization for the CUBIC model.

the number of calls to the GTR method. This magnifies the difference in computational time required by GTR and the other three methods. Additionally, in derivative-free optimization, it is expected that the vast majority of the execution time of an algorithm is spent evaluating the objective function. If minimizing a RBF model takes an amount of time comparable to evaluating the objective function, then there is little point in using RBFs. Therefore, we consider that the small increase in performance observed in Figure 6.1 is not enough to completely justify the use of GTR. We decided to keep dMS as backup solver, should GS induce too much variance in BoostGLODS in future tests.

The results for the aTPS model were identical, and so the same decisions were made.

6.2 Search step launching criteria

Regarding the search step launching criteria, three strategies were considered, MINACT, CONSEC and JUSTSUC, each with a tuning knob in the threshold required. As such, in this section we will be testing the performance of these three strategies, when applied to the initial version of BoostGLODS, but using only the RBF solvers selected in the previous section. The strategy MINACT launches the search step based on the number of active points in the list; CONSEC performs a search step based on the number of consecutive unsuccessful iterations; JUSTSUC performs a search step based on the number of unsuccessful, not necessarily consecutive, iterations. Since we wanted to have a reasonable number of search steps per run, as well as well distributed, we considered

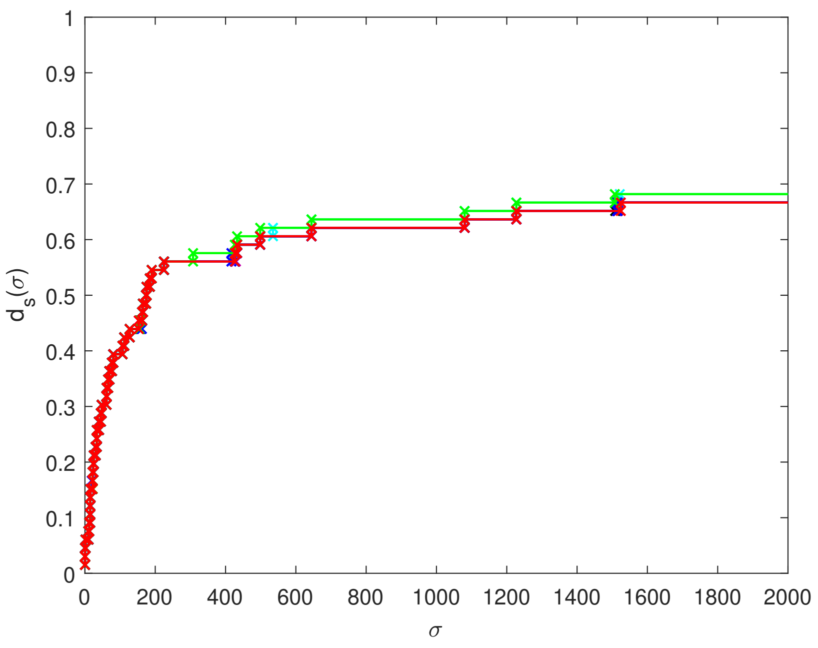

Figure 6.2: Data profiles for ten runs BoostGLODS, using the GS method to minimize RBFs (CUBIC model).

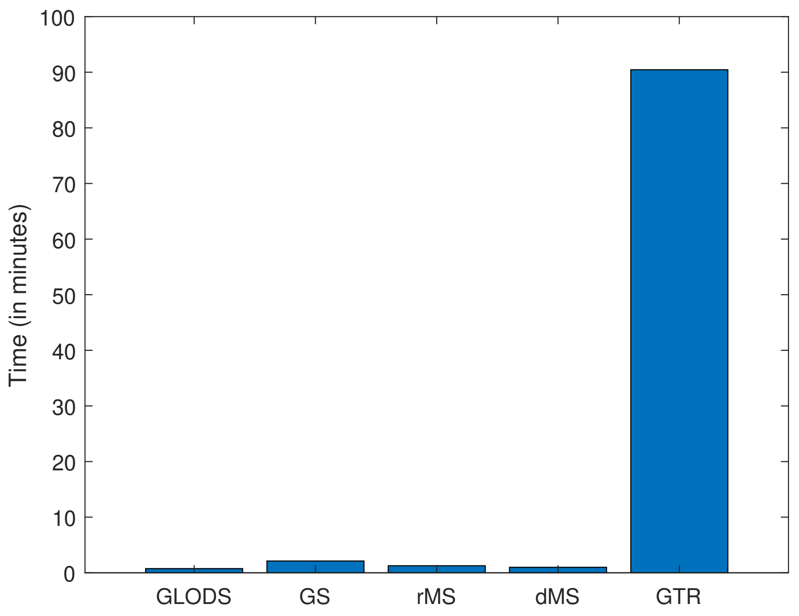

Figure 6.3: Execution time corresponding to a run of BoostGLODS when solving the problem collection.

the strategies from MINACT1 to MINACT3, CONSEC2 to CONSEC3, and JUSTSUC3. Figure 6.4 reports the scoring graphs for this batch of versions tested with the CUBIC model, where the solvers considered for minimizing RBFs are GS and dMS.

The version with the highest score is the version that uses GS coupled with CONSEC2 (search step performed when two consecutive iterations have been unsuccessful). In this version, about 30% of the iterations consist in a search step, which is much higher than the 5% previously observed. As such, it is wise to check again the variance induced by GS. This can be seen in Figure 6.5. As predicted, now that the search step is performed more often, the variance in the results is much higher than desired. Thus, we disregarded the GS method, and accepted dMS as the default solver for minimizing RBF models in BoostGLODS from now on.

Figure 6.6 displays the scoring graph for the search step launching criteria, filtering out the versions using the GS method. The best version for the CUBIC model is dMS coupled with MINACT2 (search step performed when there are two or less active points in the list not yet identified as local minima). For the aTPS model, Figure 6.7 indicates that dMS coupled with MINACT3 is the best option. It is worth noting that Figures 6.6 and 6.7 seem to indicate that the MINACT strategies selected as best for the CUBIC and aTPS models are a local optimum with regard to the associated threshold. With this purpose, we included MINACT4 in Figure 6.7.

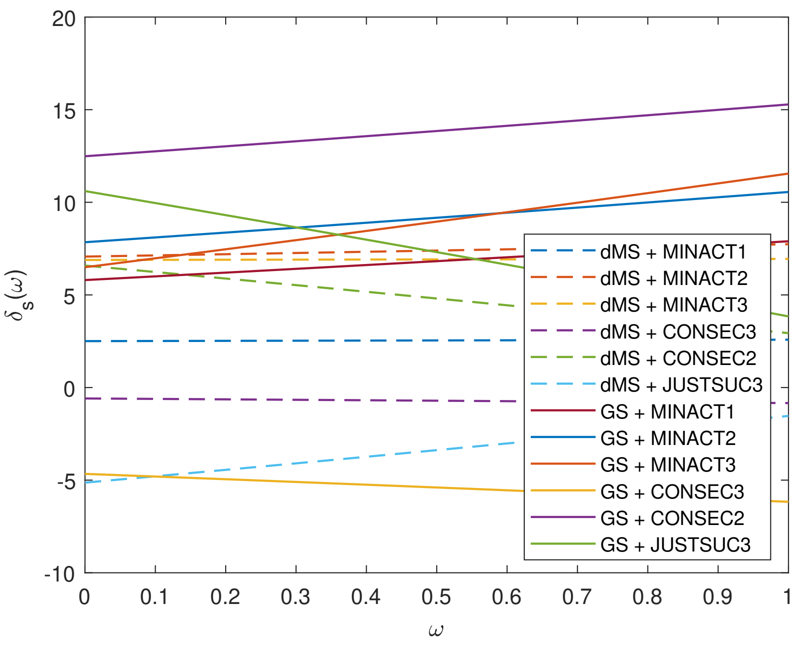

Figure 6.4: Scoring graph for the search step launching criteria, for the CUBIC model.

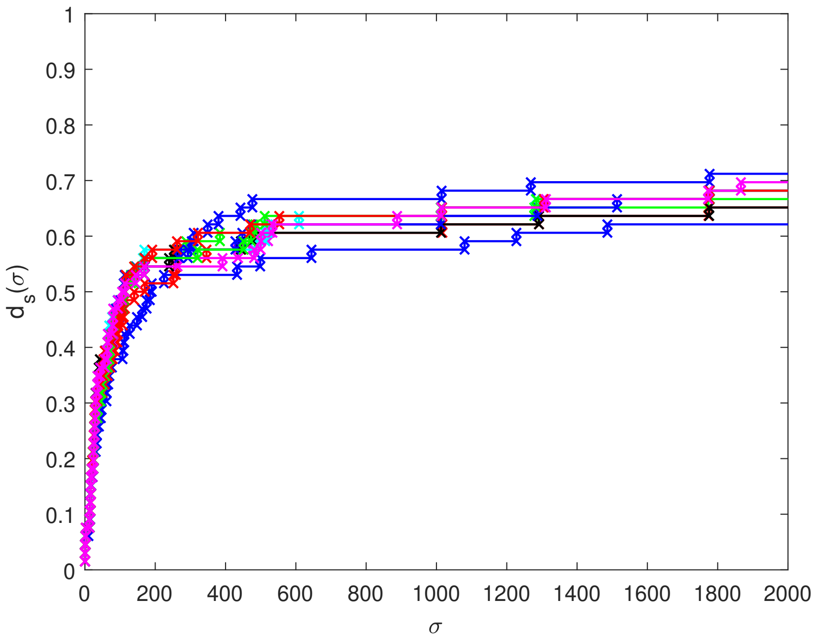

Figure 6.5: Data profiles for ten runs of BoostGLODS using the GS method to minimize RBFs and CONSEC2 to launch the search step (CUBIC model).

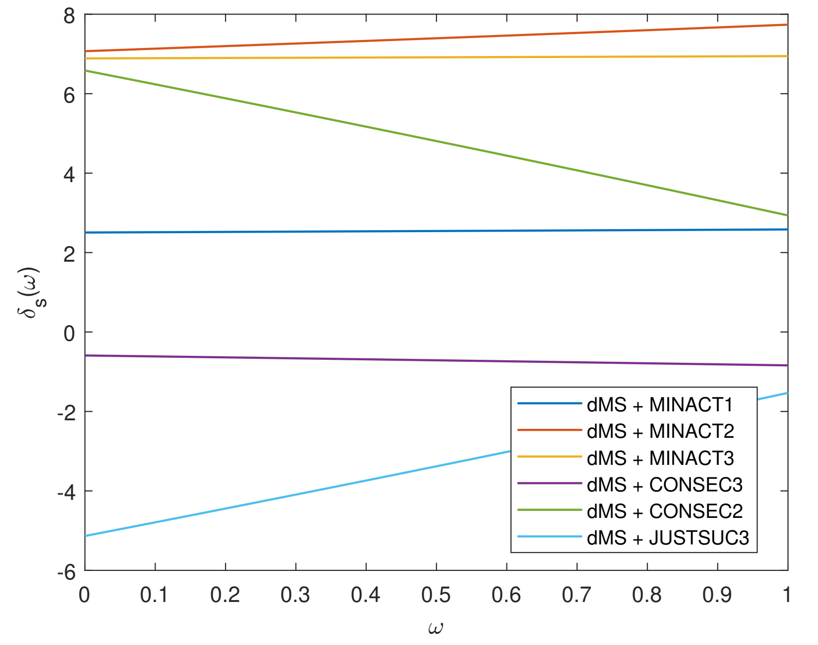

Figure 6.6: Scoring graph for the search step launching criteria, for the CUBIC model.

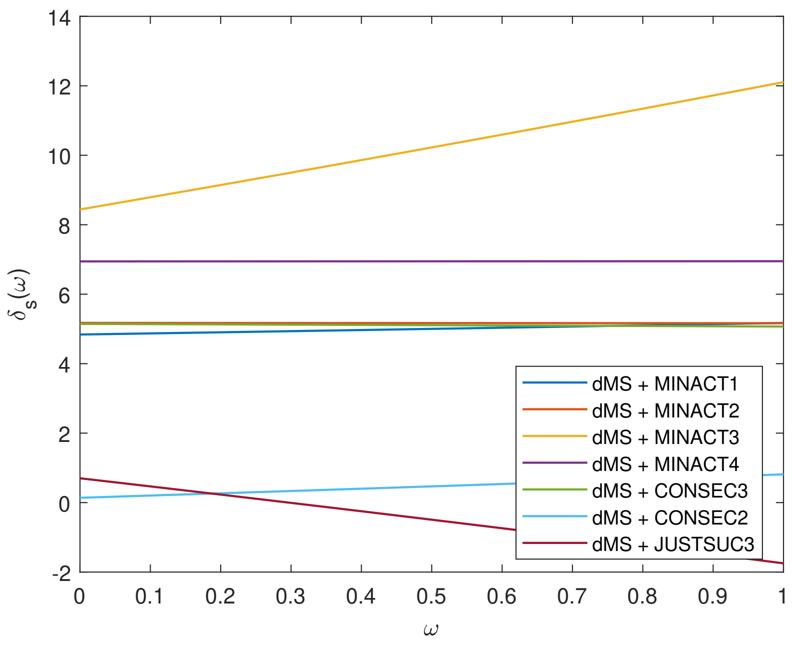

Figure 6.7: Scoring graph for the search step launching criteria, for the aTPS model.

6.3 Algorithmic flow strategies and RBF minima handling

In this section, we will test the performance of the strategies SEARCH (the search-based strategy), and NOPRIO, POINTPRIO and LINEPRIO (the poll-based strategies). Unlike the poll-based strategies, the strategy SEARCH performs a search step at every iteration, and only performs a poll step when the search step is unsuccessful. NOPRIO, POINTPRIO and LINEPRIO only perform a search step when its launching criteria are met, and perform a poll step when the search step is not performed or is performed but unsuccessful. NOPRIO picks a poll center based on only the objective function value, POINTPRIO gives priority to the points coming directly from the search step and LINEPRIO gives priority to the two best points coming from the search step as well as their successful offspring. For the SEARCH strategy, since a search step is always performed at every iteration, the search step launching criteria does not matter. For the poll-based strategies, we considered MINACT2 for the CUBIC model and MINACT3 for the aTPS model. Additionally, we selected dMS as the RBF minimizer.

Figure 6.8 displays the scores for these four strategies. It indicates that NOPRIO is the best strategy. However, we believe that the performances of POINTPRIO and LINEPRIO are close enough, so that they can be improved to be competitive against NOPRIO. Strategy SEARCH, on the other hand, is too behind for this to happen. As such, we decided not to give up on POINTPRIO and LINEPRIO yet, but disregard SEARCH.

The results for the aTPS model are identical. Thus, the same decisions were taken.

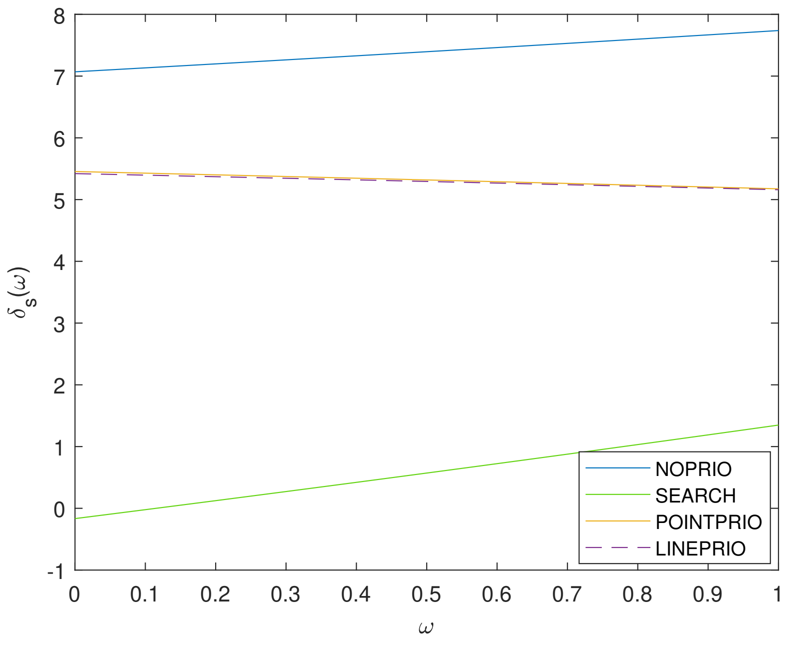

Figure 6.8: Scoring graph for the algorithmic flow strategies and RBF minima handling, for the CUBIC model.

6.4 Point selection to build RBF models

In section 3.3, we proposed three strategies that dictate which points in the list will be selected to compute RBF models: ALPHA, ACTIVE and LINES. We will combine these with the three strategies carried over from the last section. Strategy ALPHA chooses points in decreasing order of step-size, strategy ACTIVE picks active points first, and then in decreasing order of step-size, and strategy LINES chooses the starting and best points of every local line of search first, and then in decreasing order of step-size.

Figures 6.9 and 6.10 report the results for the point selection strategies for each of the algorithmic flow strategies. For strategy NOPRIO, ALPHA seems to be the best alternative, for both CUBIC and aTPS models. For the remaining combinations, the choice is not as obvious. For strategy POINTPRIO, in the CUBIC model case, ACTIVE is the best strategy. In the aTPS case, there is no major difference between the performance of the different point selection strategies. As such, we decided not to exclude any of the POINTPRIO combinations yet, for the aTPS case. For the LINEPRIO strategy, in both CUBIC and aTPS models, the results are identical regardless of the point selection strategy. However, the scores obtained with the CUBIC model are lower than the ones obtained with the aTPS model, and much lower than the ones of NOPRIO and POINTPRIO for the CUBIC model. As such, we will disregard the LINEPRIO combinations for the CUBIC model. Table 6.1 summarizes the versions that will be calibrated in the next section.

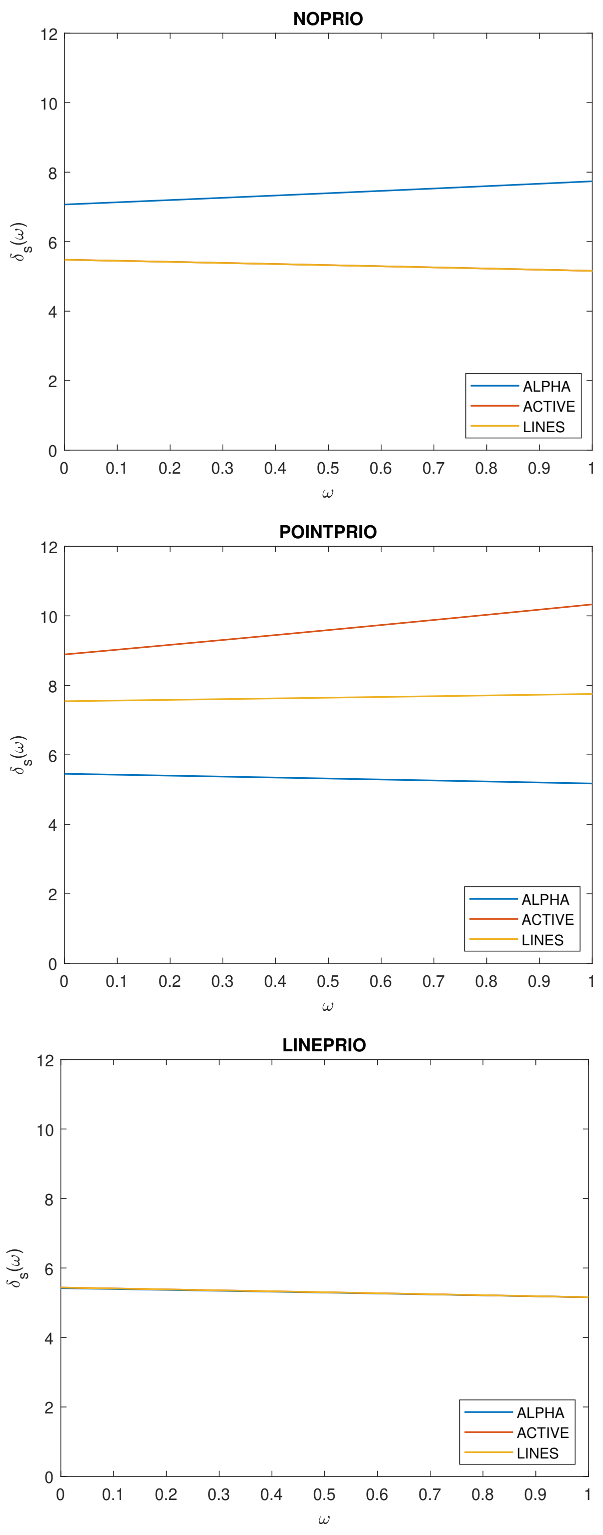

Figure 6.9: Scoring graphs for the point selection strategies, for the CUBIC model.

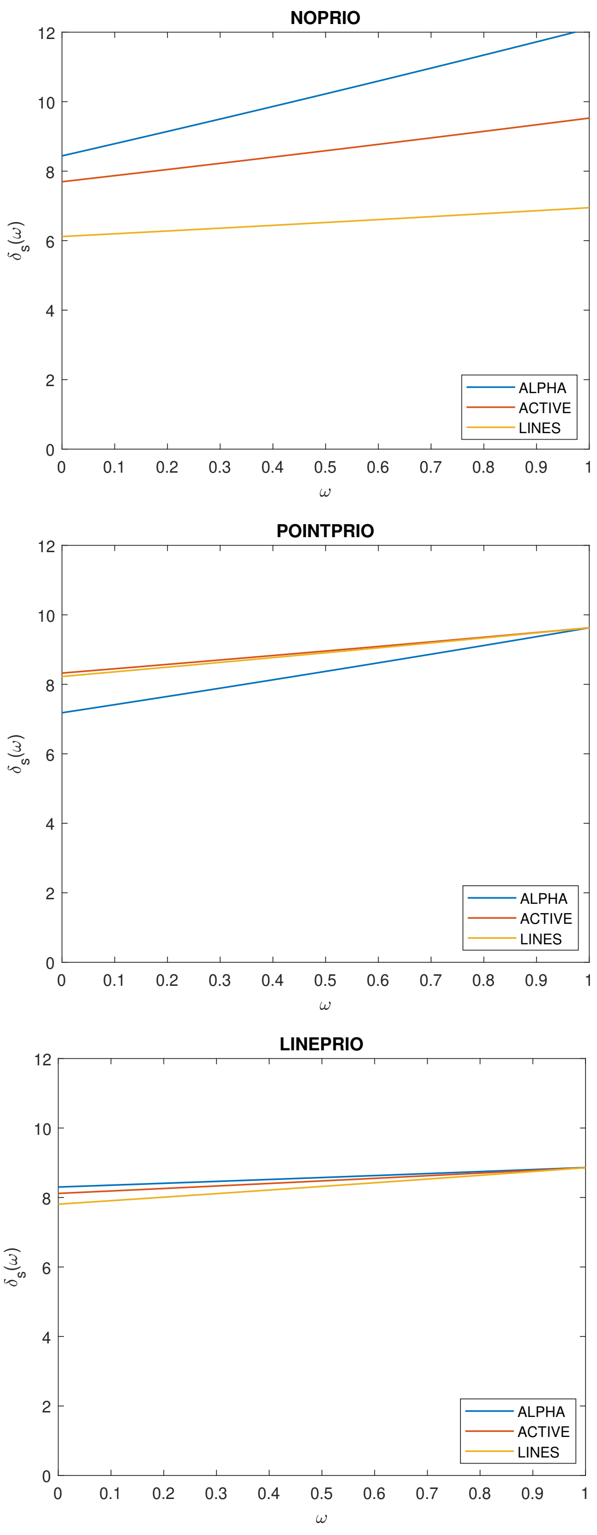

Figure 6.10: Scoring graphs for the point selection strategies, for the aTPS model.

Table 6.1: Point selection strategies for calibration.

	CUBIC	**aTPS**
NOPRIO	ALPHA	ALPHA
POINTPRIO	ACTIVE	ALPHA, ACTIVE, LINES
LINEPRIO	-	ALPHA, ACTIVE, LINES

6.5 Calibration of the number of initial points and of the upper limit to the set of points selected for RBF models

All that is left regarding the study of RBF models in BoostGLODS is to calibrate the number of points evaluated in the initialization step and the upper limit to the number of points in the set selected to compute RBF models. For the former, as we want to cover a wide range, we considered $an+b$ points, where $a \in \{1,2,3\}$ and $b \in \{0,5,10,15,20\}$. For the latter, as discussed in Section 3.3, we considered the expressions $c(n+1)(n+2)$, where $c \in \{\frac{1}{2},1,2,+\infty\}$ (QUAD, DOUBLEQUAD, QUADQUAD and INF, respectively), based on the complexity of RBF models when compared to complete quadratic models. Naturally, combining all of these options with the versions selected in Table 6.1 results in a huge number of scoring graphs to include in this work. As such, we decided to present only the versions with the best results, which can be found in Figure 6.11. Since their performance is very similar, an additional criterion is required to assess which version is the best. For this purpose, performance profiles were computed, based on the number of local minima identified for these three versions. The results are presented in Figure 6.12 and show that the aTPS version is slightly better than the other ones in both graphs, giving us confidence to select it as the best version of BoostGLODS. As such, from now on, we will refer to this version as the final version of BoostGLODS.

6.6 Quadratic tail for the aTPS model

Since the final version of BoostGLODS uses the aTPS RBF model, a quadratic tail should be implemented and tested to match the theoretical considerations outlined in Section 4.1. In fact, according to Table 3.1, the aTPS RBF model is of order 3, which requires a polynomial tail of degree 2 in order to guarantee the nonsingularity of the linear systems (3.10), built throughout the execution of the algorithm. It is worth noting that, even though the polynomial tail used in all our tests up until now is linear, we did not encounter any singular system, including the aTPS case.

Figure 6.13 displays the performance for the final version of BoostGLODS when applied a linear and a quadratic tail. Since both versions are equal in performance and the quadratic tail does not provide a noticeable increase to the performance of BoostGLODS, we kept the final version using a linear tail, as it is simpler to compute, and leads to slightly faster computations.

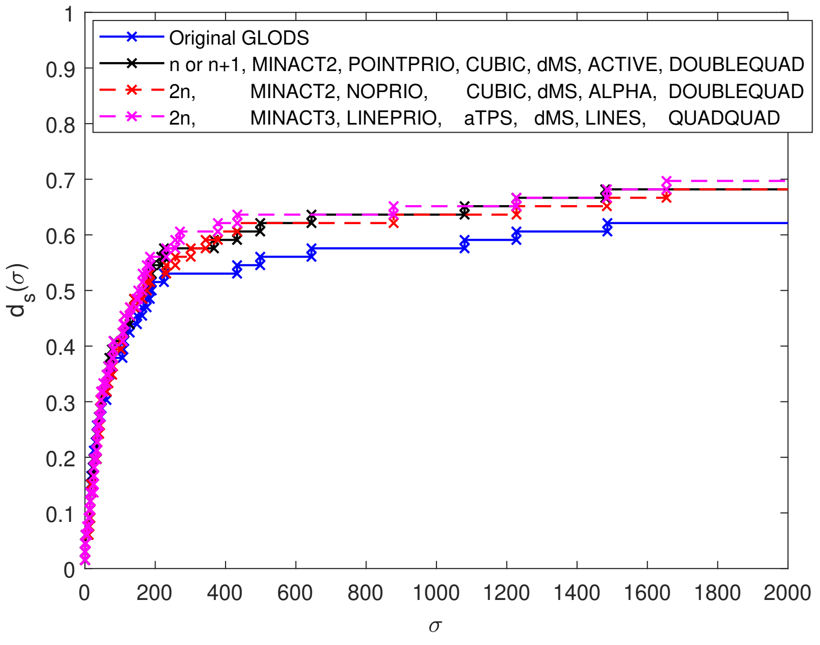

Figure 6.11: Data profiles for the best versions of BoostGLODS.

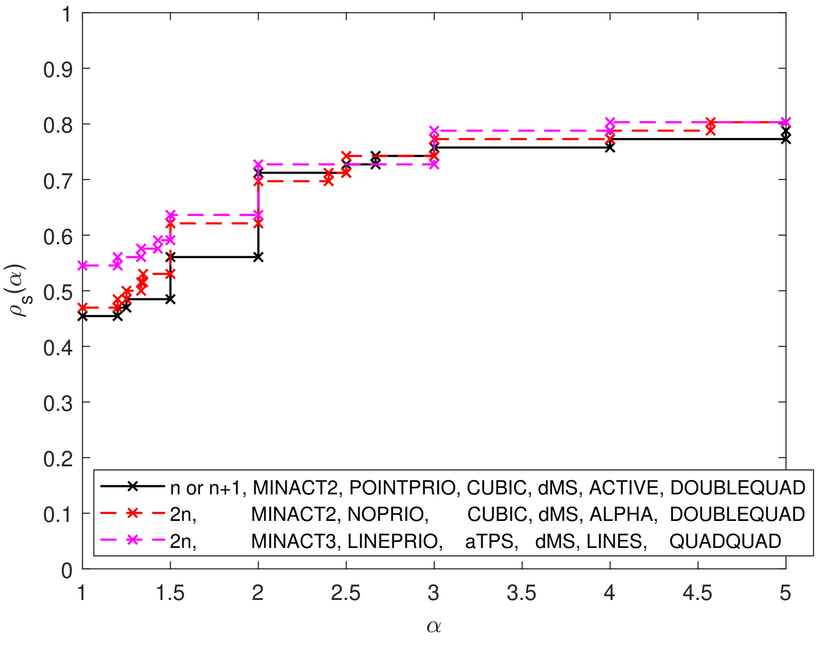

Figure 6.12: Performance profiles for the best versions of BoostGLODS, measuring the number of local minima identified.

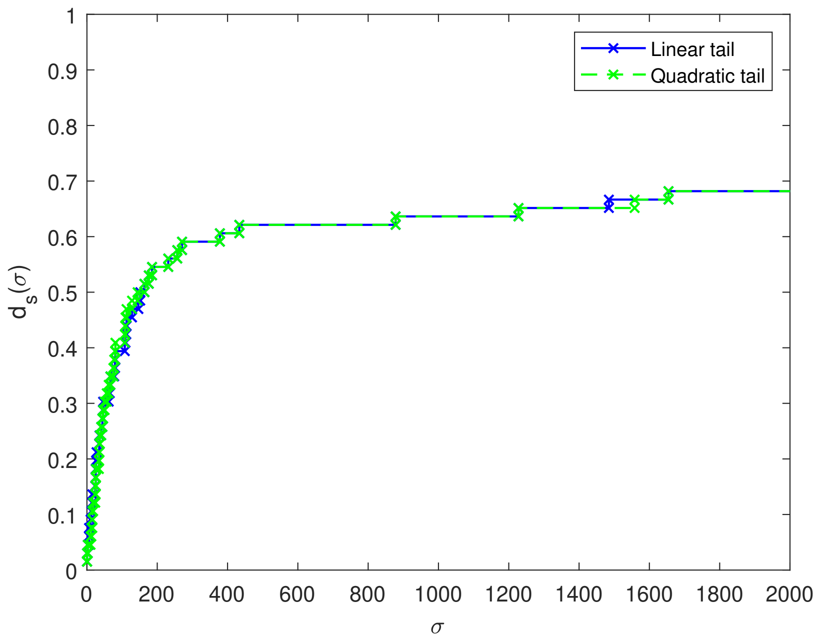

Figure 6.13: Data profiles for the linear and quadratic tails, considering the final Boost-GLODS version.

6.7 Acquisition functions

In this section, we will be testing the use of acquisition functions (ACQ) in Boost-GLODS, whose computation was detailed in Section 3.4. For the base surrogate model, we will consider the RBF models selected in the previous section. The main challenge of including ACQ functions in BoostGLODS is that they require the calibration of three extra parameters, α, δ and ϵ, that dictate how exploration of the feasible region takes place. Parameter α promotes the exploration of areas where the base RBF model might not be accurate. Parameter δ promotes the exploration of areas far away from known points. Parameter ϵ ensures that the distance function z defined in (3.15) does not vanish when the values observed in the objective function do not vary greatly.

Based on the work of Bemporad [7], to assess the performance of ACQ models in the final BoostGLODS version, we defined the following scoring problem:

$$\begin{aligned} \max \quad & \frac{\eta_{s_{(\alpha,\delta,\epsilon)}}(0.4)}{\eta_{s_{(0,0,0)}}(0.4)} \cdot 100 \\ \text{s. t.} \quad & 0 \leq \alpha \leq 3 \\ & 0 \leq \delta \leq 3 \\ & 0.1 \leq \epsilon \leq 3 \end{aligned} \tag{6.1}$$

where $s_{(\alpha,\delta,\epsilon)}$ denotes the final version of BoostGLODS when using ACQ functions with parameters α, δ and ϵ, and $\eta_{s_{(\alpha,\delta,\epsilon)}}$ denotes the scores defined in (5.6). Naturally, $s_{(0,0,0)}$

represents the final version of BoostGLODS, where the ACQ model matches the RBF model. In other words, we intend to maximize the score obtained at weight $\omega = 0.4$ for the scoring graphs comparing the final version of BoostGLODS when using only RBF models and when using ACQ models. The weight $\omega = 0.4$ was selected because the performance for low budgets is more important than for large ones. The constraints considered in this problem are the same that were considered for GLIS [7].

Considering the nature of problem (6.1), the SID-PSM [16] algorithm was used to solve it. SID-PSM is a local derivative-free optimization algorithm designed to solve constrained or unconstrained nonlinear problems. Much like GLODS, SID-PSM is also structured into poll and search steps. In the poll step, simplex gradients are estimated in order to guide local exploration, by imposing an order to the test of the poll directions. In the search step, quadratic polynomial models are built, by reusing previously evaluated points, and minimized, in an attempt of finding a better point. Regarding the initial point required by SID-PSM to solve problem (6.1), we used the values obtained in GLIS [7]:

$$\alpha = 1.5078, \quad \delta = 1.4246, \quad \epsilon = 1.0775. \tag{6.2}$$

From this starting point, SID-PSM arrived to the solution

$$\alpha = 1.50485078125, \quad \delta = 1.4246000000, \quad \epsilon = 2.5853000000. \tag{6.3}$$

Unfortunately, the excessive decimals are important because ACQ functions seem to be extremely sensitive to small variations in the parameters. For example, approximating the parameter α to 1.5049 results in a somewhat significant decrease in performance, displayed in Figure 6.14. Therefore, these models are flimsy and leave us very reluctant in using them in BoostGLODS. Moreover, ACQ models take much longer to build and evaluate than RBF models. Also, as suggested by Figure 6.15, using ACQ models does not increase the performance of BoostGLODS over just using RBF models. For all these reasons, we concluded that ACQ functions should not be used over RBFs.

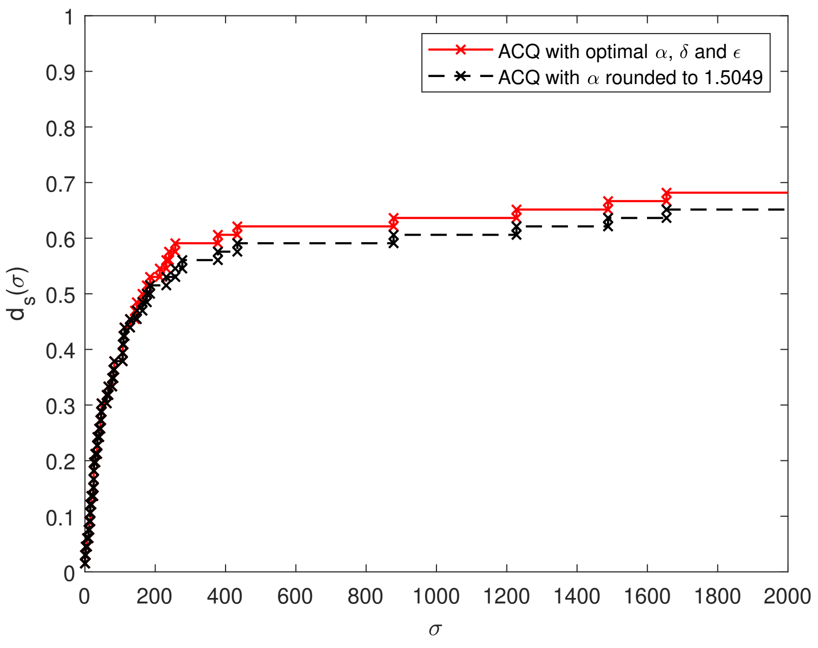

Figure 6.14: Data profiles for the ACQ versions with $\alpha = 1.50485078125$ and $\alpha = 1.5049$.

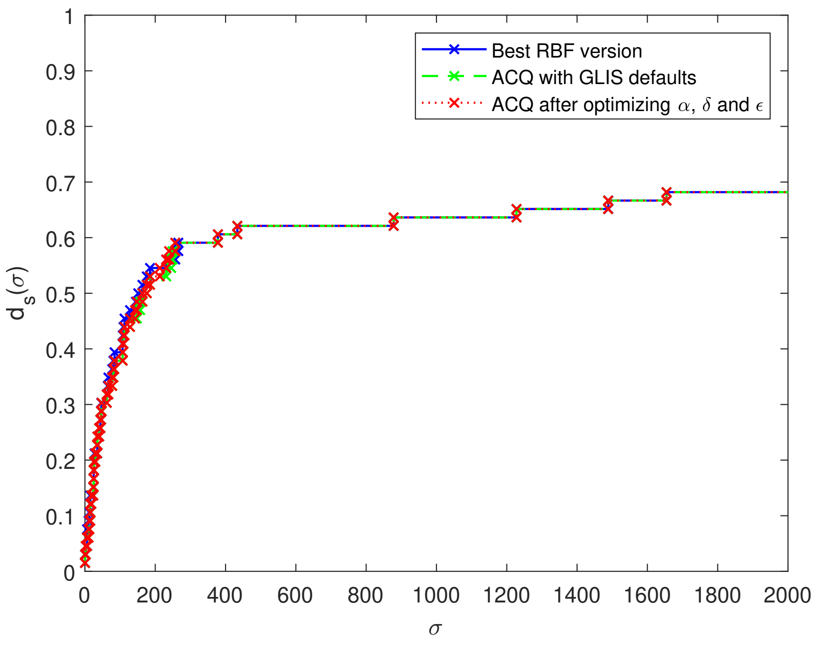

Figure 6.15: Data profiles for the ACQ tests.

7 Benchmarking BoostGLODS

In this chapter, a benchmark of BoostGLODS will be presented, using other state-of-the-art solvers in the area of global derivative-free optimization. A brief explanation of the selected solvers will be provided and their performance will be compared to the performance of BoostGLODS.

As explained in Section 5.1, the numerical assessment of the performance of Boost-GLODS was divided into two stages. This chapter corresponds to Stage 2, where Boost-GLODS will be initialized with points in a line segment, as detailed in Section 2.1.2.1, since this strategy corresponds to the best algorithmic performance of the solver, as indicated by the data profiles in Figure 7.1. Thus, to avoid bias due to the change of initialization, the translated collection of problems will be considered, as explained in Section 5.1.

7.1 Competing solvers

For the purpose of comparing the performance of BoostGLODS with other state-of-the-art global derivative-free optimization algorithms in this area of study, the following solvers were selected:

- MCS [20];
- DIRECTGL [45];
- MATSuMoTo [31];
- ZOOpt [29].

MCS [20] is an algorithm developed by Huyer and Neumaier based on partitioning the feasible region. As the algorithm progresses, the domain is divided into smaller partitions, each represented by a *pivot* point, whose objective function value is known. The domain is not partitioned uniformly; comparing different pivots guides MCS to further partition good areas of the feasible region, and leave the worse areas behind. In good areas, a local search is performed, resorting to quadratic models, in order to improve the numerical

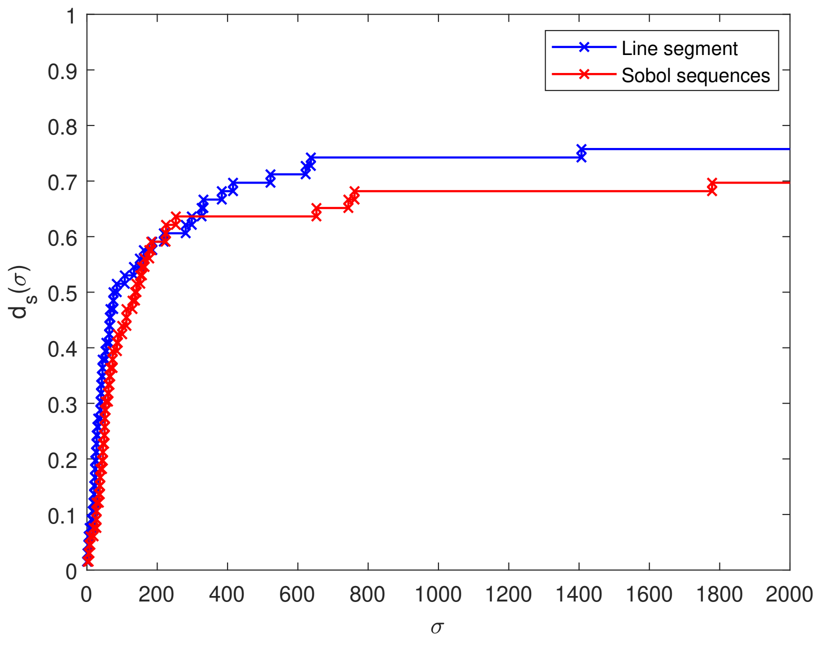

Figure 7.1: Data profiles for the final version of BoostGLODS with different initialization strategies.

efficiency of the algorithm. If the algorithm finds itself in the vicinity of a minimum, the local search is quick to identify it. Although it was released some years ago, this algorithm was selected because it presented better performance than the original GLODS and continues to be regarded as one of the best global derivative-free optimization solvers available.

DIRECTGL is part of the DIRECTGO [45] toolbox developed by Stripinis and Paulavičius, whose goal was to improve the original DIRECT [24] algorithm. DIRECT is a very popular algorithm, as it provides a way of applying Lipschitz optimization without requiring the explicit use of the Lipschitz constant. Similarly to MCS, DIRECT also works by partitioning the feasible region. However, the original implementation of DIRECT is not competitive against newer algorithms. Many attempts were made at improving its performance. DIRECTGO is a toolbox containing many of these improved DIRECT-based algorithms, DIRECTGL being one of them. Of the four algorithms with no extra input parameters presented in DIRECTGO, namely Aggressive DIRECT, DIRECTG, DIRECTL and DIRECTGL (see Table 2 in [45]), DIRECTGL is the one that presents the best performance (as suggested by Figure 7 in [45]). DIRECTGL distinguishes itself from the original implementation of DIRECT by the strategy considered to select potentially optimal hyper-rectangles (see [46]).

Of the four algorithms chosen, MATSuMoTo [31] is the only one to use a classical approach to radial basis functions and surrogate-based optimization. Developed by Müller, MATSuMoTo starts by evaluating a randomly-generated set of points. This information is then used to compute a radial basis function model of the objective function, which is minimized. The minima of the model are evaluated in the true objective function and added to the model, to increase its accuracy. Other candidate points may be considered for evaluation in addition to or instead of the surrogate minima, based on a score. This process repeats itself until a stopping criterion is met. MATSuMoTo is guaranteed to find the global minimum, if an infinite number of evaluations is allowed, as result of a restarting strategy. When MATSuMoTo recognizes that no progress is being made (the algorithm has converged to a local minimum), it restarts from scratch, quarantining all points found so far, excluding them from being considered in the computation of any radial basis function model in posterior iterations. MATSuMoTo is a natural choice as comparison solver, since it is a very good example of applying radial basis functions to global derivative-free optimization.

ZOOpt [29] is an algorithm based on classification techniques. Much like MCS and DIRECTGL, it also works by dividing the feasible region. However, unlike the previous two algorithms, the partition is not kept nor does it evolve from iteration to iteration. As such, division begins from scratch in every iteration. In the first iteration, a set of randomly generated points is evaluated, and each point is classified as good or bad. Most commonly, only the best point (of lowest function value) is classified as good. Points that are good are carried over to the next iteration, and the division process begins. For each variable and good points, a randomly-generated range is selected, as to not violate the bounds of the problem. This results in the good points each being encapsulated in a hyper-rectangle. Then, sampling is performed in the areas strictly outside any of the hyper-rectangles generated. The sampled points are then evaluated in the objective function, all points are reclassified, and this process begins anew.

7.2 Performance comparison

Since we are comparing BoostGLODS to algorithms with random components, data profiles, as defined in Section 5.2, are not enough. As such, we decided to alter how data profiles are constructed, in order to accommodate randomness. Let q be a random solver and d_{q_i}, with $i \in \{1,\dots,m\}$, the data profiles constructed for each individual run when comparing q to other solvers. Based on d_{q_i}, we built two extra data profiles, in order to provide a range of performance for solver q at every budget level σ,

$$d_{q_{best}}(\sigma) = \max_{i\in\{1,\dots,m\}} d_{q_i}(\sigma), \ \ \forall\sigma \in [0, 2000], \tag{7.1}$$

$$d_{q_{worst}}(\sigma) = \min_{i\in\{1,\dots,m\}} d_{q_i}(\sigma), \ \ \forall\sigma \in [0, 2000], \tag{7.2}$$

where $d_{q_{best}}$ indicates the upper limit of performance for solver q over m runs, and $d_{q_{worst}}$ the lower limit, for every budget σ.

In addition to these two new data profiles, another one can be constructed, in order to indicate where the performance of q over m runs will most often be located. For every

$\sigma \in [0, 2000]$, without loss of generality, order $d_{q_i}(\sigma)$, $i \in \{1, \ldots, m\}$ such that $d_{q_1}(\sigma) \leq \ldots \leq d_{q_m}(\sigma)$. Then,

$$d_{q_{med}}(\sigma) = \begin{cases} d_{q_{\lceil \frac{m}{2} \rceil}}(\sigma), & m = 2R-1 \\ \frac{d_{q_{\frac{m}{2}}}(\sigma)+d_{q_{\frac{m}{2}+1}}(\sigma)}{2}, & m = 2R \end{cases}, \quad \forall \sigma \in [0, 2000], \ R \in \mathbb{N}. \tag{7.3}$$

In other words, $d_{q_{med}}(\sigma)$ represents the median performance of q over m runs at every budget σ. Therefore, we can replace the original data profiles d_{q_i} with the new profiles $d_{q_{best}}$, $d_{q_{worst}}$, and $d_{q_{med}}$. This way, we have a tool that is capable of comparing the performance of random solvers against deterministic ones. This procedure was applied to MATSuMoTo and ZOOpt, after running them ten times each over the translated collection of problems. The results are displayed in Figure 7.2.

At the precision level of $\tau = 10^{-5}$, unfortunately BoostGLODS is not able to surpass MCS, which stands in the lead by a significant margin. However, the performance increase observed in BoostGLODS by comparison with the original GLODS is very noticeable. In fact, this increase allows BoostGLODS to present a better performance than MATSuMoTo at every budget, even if by a small margin for low budgets. GLODS was only as good as MATSuMoTo's worst case scenario, for low budgets, and would perform similarly to MATSuMoTo's median case, for larger ones. Also due to this increase in performance, the budget threshold at which DIRECTGL surpasses GLODS went from $\sigma = 700$ (when comparing to the original GLODS) to $\sigma = 1500$ (when comparing to BoostGLODS), meaning that BoostGLODS remains relevant for much larger budgets. In fact, DIRECTGL's growth in percentage of problems solved is relatively steady through the budget, whereas BoostGLODS presents a very rapid growth until $\sigma = 300$, slowing down significantly afterwards, which makes it significantly better than DIRECTGL for lower and medium budgets of function evaluations. Moreover, the gap in performance between BoostGLODS and DIRECTGL is much larger at low and medium budgets (where BoostGLODS is better) than at high budgets (where DIRECTGL is better).

For low levels of precision ($\tau = 10^{-3}$), all solvers are much closer in performance, with MATSuMoTo pulling slightly ahead of its competitors. BoostGLODS is one of the solvers at the higher end of performance for all budget levels, whereas GLODS is on the lower end. MATSuMoTo's lead can be explained by the fact that it is a very good algorithm to find good areas of the feasible region. By spraying a number of points across the feasible region, and aided by RBF models, MATSuMoTo quickly identifies promising areas. However, it does not possess good capabilities to refine points identified as good, leading to a decline in performance relative to other solvers as the precision level increases. On the other hand, algorithms such as MCS, BoostGLODS, GLODS, and DIRECTGL do not suffer great loss in performance. For $\tau = 10^{-7}$, DIRECTGL surpasses GLODS at medium budgets, but never surpasses BoostGLODS. Moreover, the gap between MCS and BoostGLODS is much narrower than for $\tau = 10^{-5}$. Thus, we can conclude that BoostGLODS is very competitive, being only outperformed at this stage by MCS.

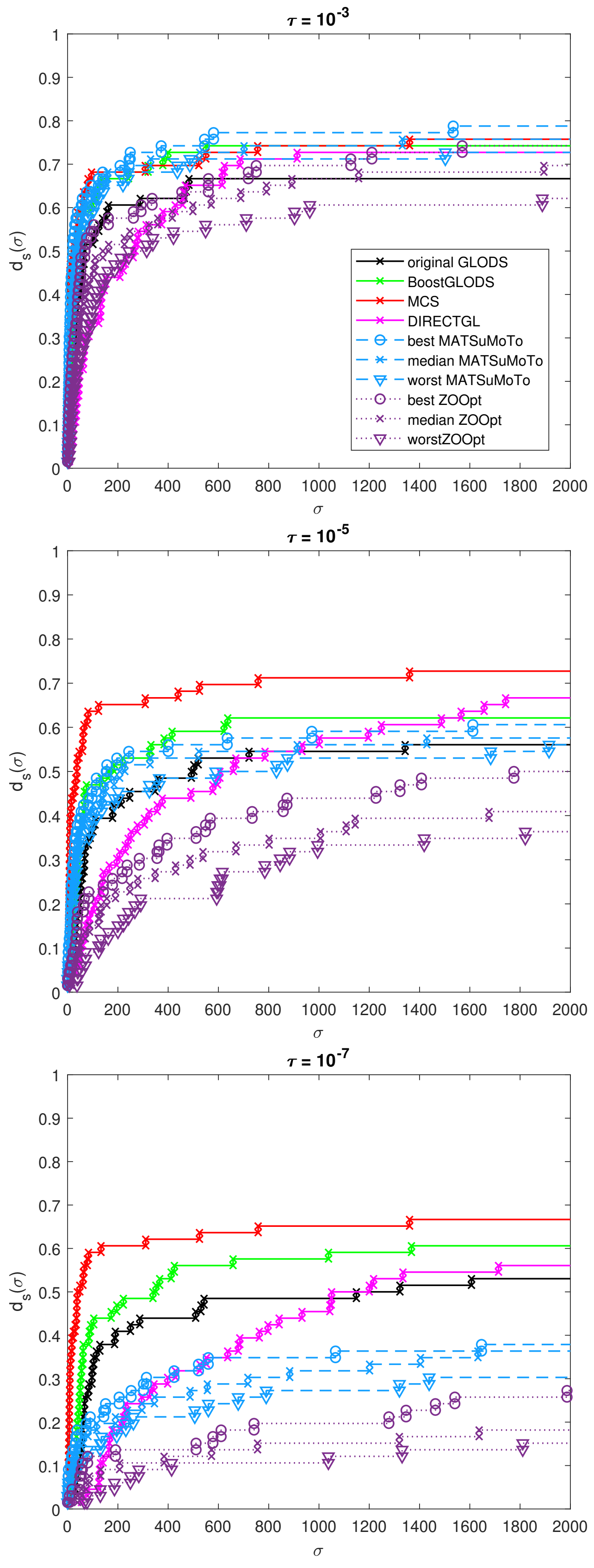

Figure 7.2: Data profiles for BoostGLODS and its competitors.

8

Parallelization of BoostGLODS

In a context of expensive function evaluation, each function value computed at the search or poll steps directly contributes to the performance of the algorithm. However, in BoostGLODS, all these points are queued and evaluated sequentially, one at a time. For this reason, BoostGLODS lends itself well to parallel strategies.

In this chapter we will use parallel strategies to simultaneously evaluate multiple points, in each step of BoostGLODS. This should result in a significant decrease in the execution time of the algorithm for expensive functions, as multiple calls to the objective function, that previously were executed in a queued fashion, are now condensed into the same time frame. Figure 8.1 illustrates this idea, considering the evaluation of five points.

In the next sections, we will detail the strategies implemented for the parallelization of BoostGLODS, and we will comment on the results obtained. It is important to note that all experiments in this chapter considered the conditions described for Stage 1, regarding the collection of problems and the initialization strategy for BoostGLODS.

8.1 Parallel strategies

In order to discuss the introduction of parallel strategies in BoostGLODS, two fundamental concepts are necessary. The first is the concept of *worker*. A worker represents an entity responsible for evaluating the objective function at a given point. Each worker may only evaluate one point at a time, but multiple workers may be used simultaneously. The more workers available at a time, the more points BoostGLODS may evaluate at once. Workers are independent from each other, unless strictly stated otherwise. Figure 8.1 also illustrates the use of multiple workers: in the sequential version, only one worker is used, resulting in all points being queued and evaluated in sequence; in the parallel version, five workers are used, resulting in all points being evaluated simultaneously. The second concept essential to parallelism is the concept of *batch*. A batch is a set of points to be simultaneously evaluated within a step, taking into consideration the number of workers. In other words, a batch must fulfill the following condition

$$|B| \leq w, \tag{8.1}$$

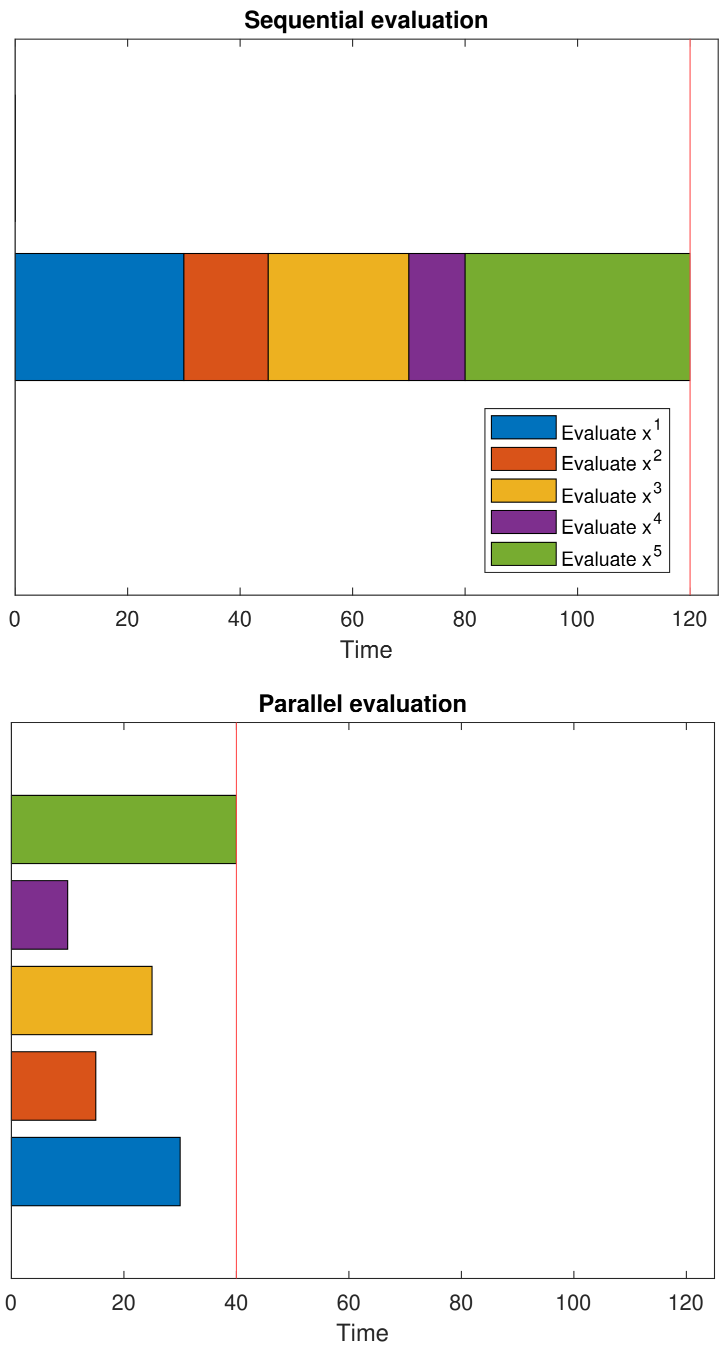

Figure 8.1: Execution time of sequential and parallel strategies.

where B is the batch and w is the number of workers available. A batch cannot contain points from different steps of the algorithm. The final version of BoostGLODS, calibrated in Chapter 6, evaluates $2n$ points in the initialization step. Since BoostGLODS uses opportunistic coordinate search, each poll step evaluates at most $2n$ points. As the MultiStart method used in the final version of BoostGLODS is initialized with a set of n points, then at most n points result from it. Also, when using Sobol sequences, n points were generated. In other words, each search step evaluates at most n points. As such, the number of batches required to evaluate all points in the initialization or poll steps is

$$\left\lceil \frac{2n}{w} \right\rceil, \tag{8.2}$$

and in the search step is

$$\left\lceil \frac{n}{w} \right\rceil. \tag{8.3}$$

For example, suppose that there are 4 workers available. To solve the *becker_lago* problem of dimension 2, only one batch is needed per step of the algorithm, as indicated by conditions (8.1), (8.2), and (8.3). Consider now the *ackley* problem of dimension 10, with the same amount of workers available. Now, up to 20 points may be evaluated in the poll step. However, since only 4 workers are available, only 4 points may be evaluated simultaneously. As such, in this case, the points to be evaluated in the poll step are split into 5 batches (see (8.2)), so that each individual batch fulfills condition (8.1). In the search step, since 10 points are to be evaluated at most, 3 batches are required (see (8.3)). The concept of batch is very useful in parallelization because one batch in a parallel algorithm can be considered of similar cost to one function evaluation in a sequential one.

The first strategy for parallelization is the natural one, consisting in directly replacing the queue system in each point-evaluating step of BoostGLODS (initialization, poll, and search steps) with the worker-based system, allowing for parallel evaluation of multiple points in the objective function. Both the order in which the points are evaluated, as well as the opportunistic nature of BoostGLODS in the poll step (once a point is successfully added to the list, all other evaluations within the step are canceled), were kept, so that the sequential version of BoostGLODS selected in Chapter 6 matches the parallel version of BoostGLODS, when only one worker is available.

The second strategy for parallelization consists in selecting more than one poll center in each poll step, leading to the additional consideration of which points should be selected. For this purpose, two approaches were considered. The first is letting BoostGLODS select points naturally, based on the point choice strategy being used (NOPRIO, POINTPRIO, or LINEPRIO). For example, for the NOPRIO strategy, instead of considering only the active point with the lowest objective function value, we can also select the active point with the second-lowest objective function value, totaling two poll centers. A second approach consists in forcing BoostGLODS to select other additional points for polling. In this case, BoostGLODS always selects the best active point in the list, not yet identified as a minimum, as a poll center, before selecting other points based on the point choice strategy.

Since we can now force BoostGLODS to always select the best point as a poll center and still select additional poll centers, we considered an additional point choice strategy, based on the step-sizes of the points in the list. This strategy is called **ALPHAPRIO** and selects as poll centers points in decreasing order of step-size, and in increasing order of objective function value, in case of ties. This strategy is interesting because, coupled with forcing the selection of the best active point as a poll center, it allows BoostGLODS to simultaneously refine the best points in the list and explore good areas of the feasible region.

In addition to which points to select, it is also a concern how many poll centers should be considered. For this purpose, a number of poll centers may be defined by the user, or we can allow BoostGLODS to dynamically determine how many poll centers can be selected for each problem, based on the number of workers available, problem dimension and/or number of active points in the list. In this work, if the number of poll centers is defined by the user, meaning that it is fixed, we considered two or three poll centers. When BoostGLODS dynamically decides how many poll centers should be selected, since evaluating the offspring of a poll center requires at most $2n$ evaluations, we considered the number of poll centers given by

$$\max\left\{\left\lfloor\frac{w}{2n}\right\rfloor, 1\right\}, \tag{8.4}$$

where n is the problem dimension, and w is the number of workers available. In Section 8.2, this strategy will be denoted by *per-problem poll centers*. Finally, we also considered a strategy where all active points are selected as poll centers. Naturally, for this strategy, no considerations on type or number of points to be selected are required, since only active points not yet identified as local minima can be selected as poll centers.

8.2 Computational results

The first results we will discuss relate to simply replacing the queue system in the final version of BoostGLODS, calibrated in Chapter 6, with the worker-based system. Since data profiles were not designed with parallel algorithms in mind, some modifications were required. In the original data profiles, function $h_{p,s}$ measures the number of function evaluations until problem p is considered solved by algorithm s. In this first experiment of parallelization, *modified data profiles* were considered, where function $h_{p,s}$ measures the number of batches required. Results are displayed in Figures 8.2 and 8.3, and correspond to the first strategy discussed in the previous section.

As expected, Figure 8.2 indicates that the availability of more workers leads to better performance. In fact, as the number of workers increases, the number of points allowed in each batch also increases (see condition (8.1)), resulting in less batches needed to solve the same problem. However, there are significant diminishing returns as the number

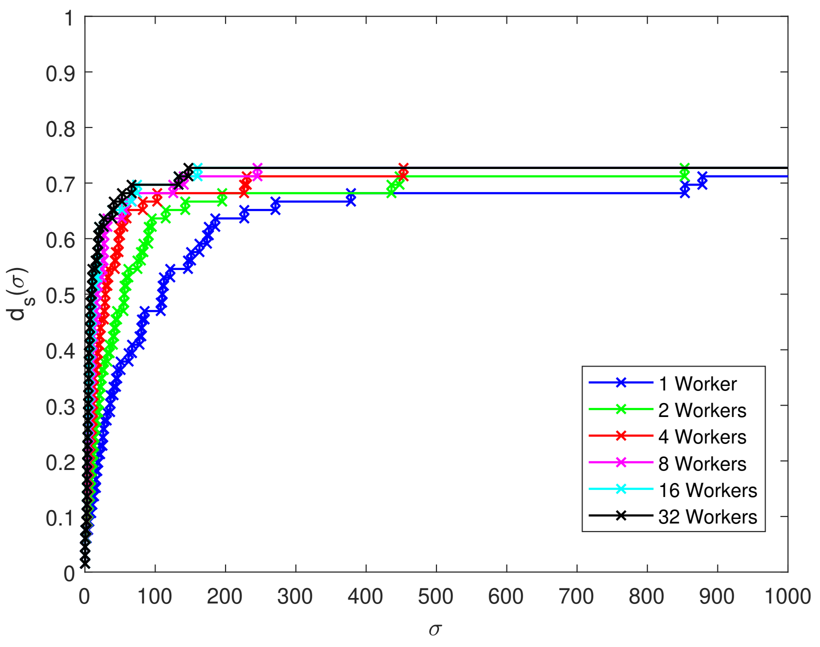

Figure 8.2: Modified data profiles for the parallel BoostGLODS versions, corresponding to different numbers of workers.

of workers increases. This is due to the distribution of the dimension of problems in the collection. For example, increasing the number of workers from 2 to 4 provides a benefit to all problems. The lowest problem dimension is 2. Thus, the number of batches required in every poll step goes from 2 to 1. However, increasing the number of workers from 8 to 16 does not provide a benefit to all problems in the collection. For example, with 8 workers available, for any problem with dimension 4 or lower, the poll step already only required 1 batch. Increasing the number of workers to 16 does not lower the number of required batches. As such, benefits are only obtained for problems of dimension 5 or higher.

The performance profiles reported in Figure 8.3 respect to computational time, computed as

$$t_{p,s} = e + m \cdot u, \tag{8.5}$$

where e is the elapsed time between starting and stopping the algorithm, m is the number of batches evaluated until the algorithm stopped, and u is the estimated time for one objective function evaluation. This metric provides an estimate of the total time required by the algorithm if one objective function evaluation took u seconds. It is important to note that, for real cases with expensive objective functions, it is usual that one evaluation takes one or more seconds to be completed. However, in Figure 8.3, the benefits of parallel strategies in BoostGLODS can already be seen for values of u as low as 0.1 seconds. On the other hand, for lower values of u such as 0.01 and 0.001, using parallel strategies is worse than the sequential version of BoostGLODS. This is the result of the large overhead

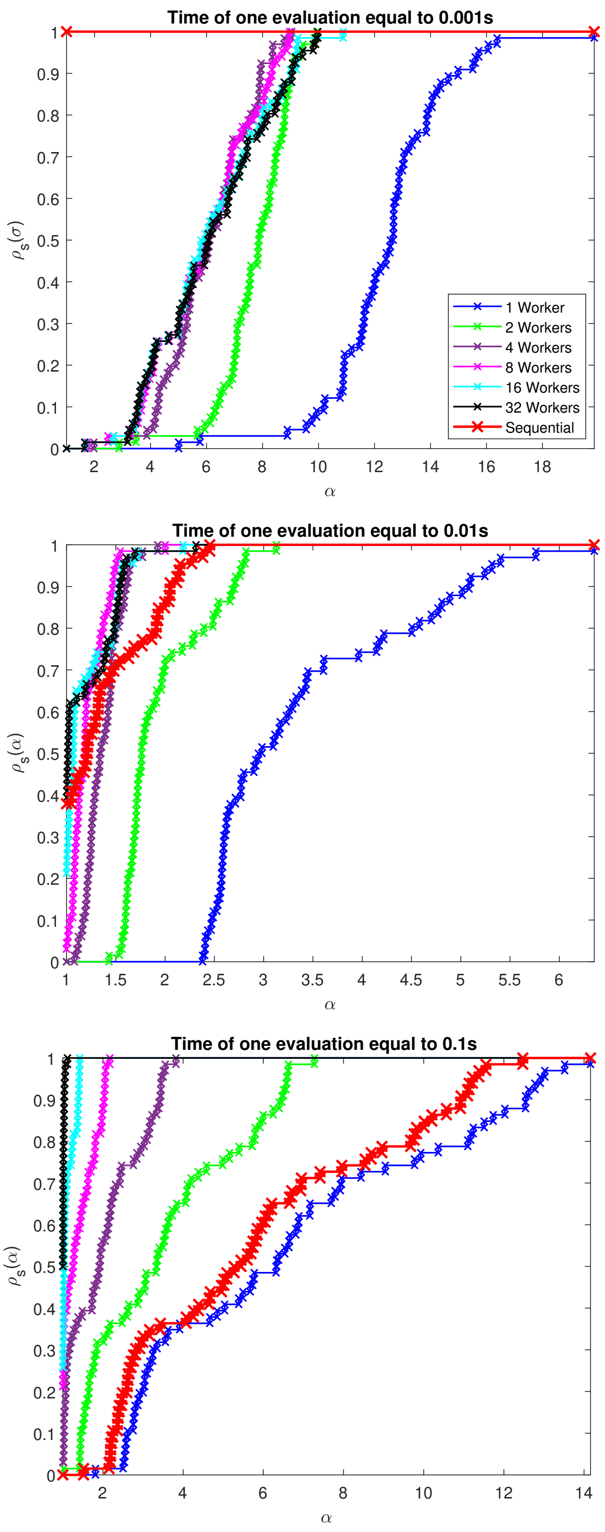

Figure 8.3: Performance profiles for the execution time of the parallel BoostGLODS versions, corresponding to different numbers of workers.

associated with the worker-based system, specially noticeable in the performance profile corresponding to $u = 0.001$ seconds (see Figure 8.3). In fact, in this case, the sequential version and the version with one worker evaluate the same exact points in the same exact order; the only difference is the system used. However, these values for u are not relevant in global derivative-free optimization.

Finally, the results for the strategies regarding the selection of additional poll centers are displayed in Figure 8.4. Given the previous results, these strategies were tested using the worker-based system instead of the sequential queue-based one, with 32 workers available. In each data profile, there is a version which only selects one poll center in each poll step. For the LINEPRIO data profile, this is the worker-based implementation of the final version of BoostGLODS, calibrated in Chapter 6. For the ALPHAPRIO data profile, it is the same as the LINEPRIO one, but replacing the strategy LINEPRIO with the strategy ALPHAPRIO. For the POINTPRIO data profile, it is the worker-based implementation of the version of BoostGLODS in Figure 6.11 that uses the POINTPRIO strategy. In all three data profiles, the versions that select more than one poll center use the same settings as the respective version that selects only one. Each version is identified by the number of poll centers selected in each poll step, when it is fixed and defined by the user, or by *per-problem* or *all active points as*, when it is dynamically computed. *Per-problem* explicitly calculates the number of poll centers for each problem, according to (8.4), and *all active points as* simply selects all active points not yet identified as local minima as poll centers. In the absence of *with best point prio*, the corresponding version selects poll centers naturally as indicated by its respective strategy (LINEPRIO, ALPHAPRIO, or POINTPRIO). Otherwise, the best point is always selected as a poll center.

Unfortunately, the results indicate that selecting more than one poll center is detrimental to the performance of the algorithm. In Figure 8.4, regular data profiles are presented, that show that all versions tested solve less problems than the version that only selects one poll center. This could be explained by the fact that selecting additional poll centers also forces the algorithm to spend more function evaluations to reach the same outcome as the version that selects only one poll center. However, benefits can still be obtained regarding computational time, as a version that selects more than one poll center will make better use of the workers available than one that only selects one poll center. For example, suppose 32 workers are available. Then, for the *becker_lago* problem of dimension 2, a version that only selects one poll center may evaluate up to 4 points simultaneously, which only uses 4 of the 32 available workers. On the other hand, a version that selects two poll centers may evaluate up to 8 points in the poll step, which can all still be evaluated simultaneously, due to the number of workers available. This means that the version that selects two poll centers evaluates up to twice as many points in the same time frame. This could make the versions that select additional poll centers reach the global minimum faster, despite the results reported in Figure 8.4. However, this does not seem to be the case. In Figure 8.5, modified data profiles for the same LINEPRIO versions as in Figure 8.4 are displayed. In these data profiles, the performance is now assessed in terms of the number of batches required to solve a problem, as opposed to the number of function evaluations. Each batch may only contain points within the same step, but may contain points from different poll centers, as long as the poll centers are selected in the same poll step. This means that, in the scenario described earlier considering the *becker_lago* example, a batch corresponds to up to 4 function evaluations for the version that selects one poll center, and up to 8 for the version that selects two. In other words, assessing performance in terms of batches takes into account the difference in worker availability usage. Even in this case, the versions that evaluate more than one poll center show worse performance.

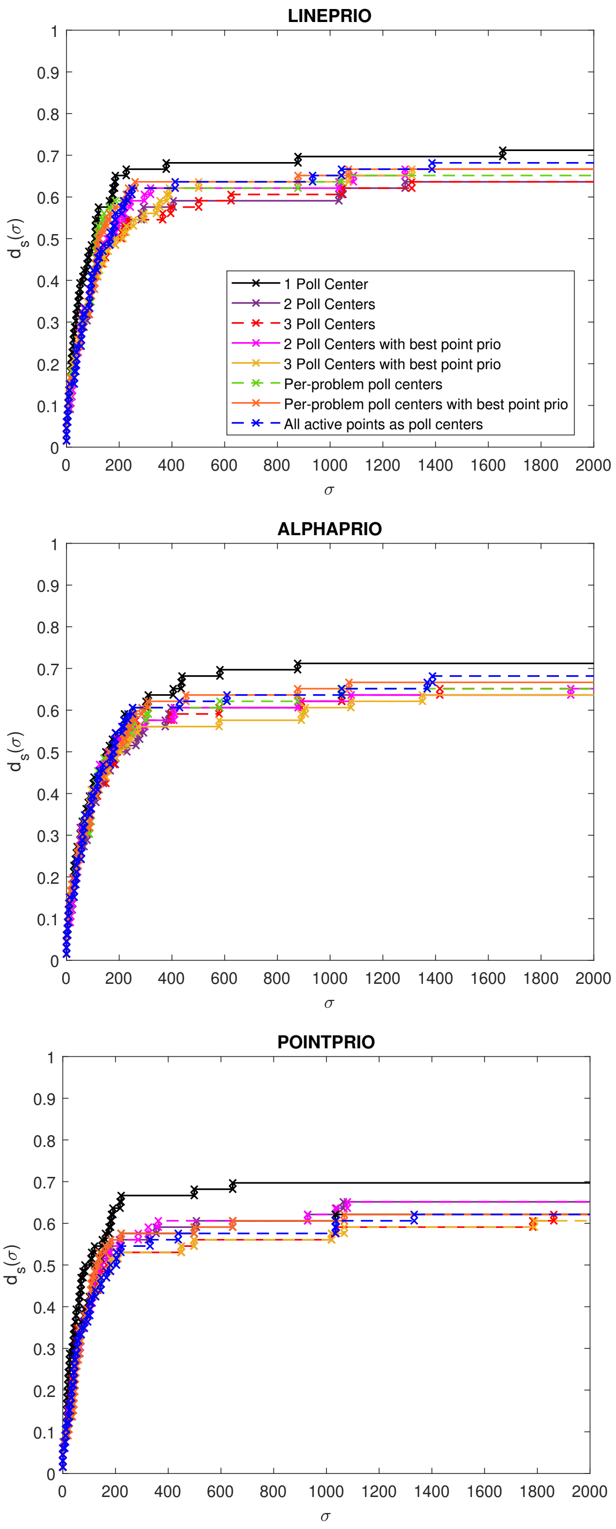

Figure 8.4: Data profiles for the selection of additional poll centers.

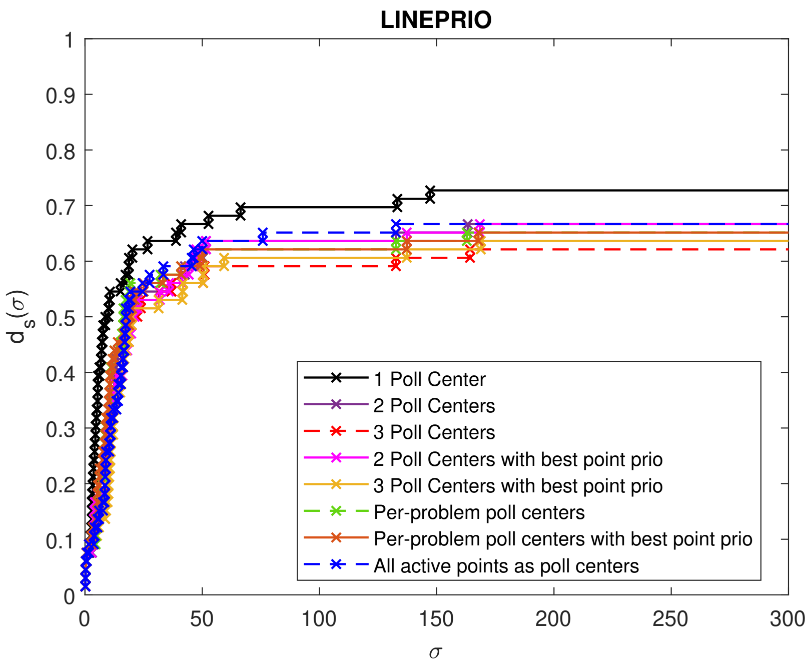

Figure 8.5: Modified data profiles for the selection of additional poll centers.

9 Conclusions and open questions

At the time of its first release, GLODS was a competitive derivative-free optimization solver, based on directional direct search, using pseudo-random techniques to initialize new lines of search, but not taking advantage in this initialization process of the information gathered by the algorithm in previous iterations. In this work, radial basis functions were proposed to define a search step in GLODS, at no extra cost in terms of function evaluations, since models are computed by reusing points previously evaluated.

Questions like which family of radial basis functions to consider, as several different expressions are available, which sets of points to use to compute these global models, as the corresponding quality directly relates to the set of points selected, and how to minimize the computed models were addressed, as well as how to incorporate these models in the algorithmic definition of GLODS, by defining criteria for how often a search step should be performed and how the different minima of the radial basis functions should be explored.

After extensive numerical testing of the different algorithmic variants considered, translating different answers to the previous questions, the best version of GLODS, now entitled BoostGLODS, uses Thin-Plate Splines radial basis functions, computed by selecting up to $2(n+1)(n+2)$ points to build the models, based on the history of each line of search considered by the algorithm. Models are minimized with a deterministic version of MultiStart, initialized using n of the points selected to build the models, and the corresponding minima are evaluated in the true objective function. The two best points found, and their successful offspring, are further explored in the poll step of the algorithm. This search step is performed once there are three or less active points in the list of points evaluated, not yet identified as local minima. Results indicate that this version clearly outperforms the original GLODS, in terms of numerical performance.

Additionally, BoostGLODS was compared with other state-of-the-art global derivative-free optimization solvers, namely MCS, DIRECTGL, MATSuMoTo, and ZOOpt, some of which present a random behavior. Results indicate that the new algorithm is extremely competitive with these solvers. With exception of MCS, BoostGLODS presents a better numerical performance than all the solvers tested, regardless of the precision level considered for solving the problems and of the budget of function evaluations allowed. In what respects to MCS, for low precision values, BoostGLODS is competitive. Due to the randomness presented in some of the algorithms tested and to the large number of comparisons that needed to be performed, data profiles were adapted to address random behavior of algorithms and a new type of performance tool was proposed, namely the scoring graphs.

Finally, parallel strategies were implemented and tested. Again, data profiles were adapted to measure the performance of parallel algorithms, by assessing it in terms of number of batches required to solve a problem, as opposed to number of function evaluations.

Parallel variants comprised the simple parallel evaluation of function values at the different steps of BoostGLODS, to more sophisticated strategies that simultaneously selected different poll centers to be evaluated in parallel. As expected, results suggest that the parallel evaluation of objective functions values is advantageous for average function evaluation times as low as 0.1 seconds. Strategies that simultaneously selected more than one poll center indicated a decrease in the performance of the algorithm.

Despite the thorough study performed on how to incorporate global models in GLODS, based on radial basis functions, alternatives could have been considered. Namely, Gaussian processes are another type of surrogate models used to capture the global behavior of a function. Even in the radial basis class of functions, there are many families that could have been explored. The Cubic and Thin-Plate Splines expressions were considered due to their easy implementation, since no additional parameters are required in the corresponding definition. However, other options could have been taken. In the near future, BoostGLODS will be incorporated in the derivative-free optimization toolbox BoostDFO [2], freely available to the research and industrial communities, allowing an efficient and easy use of global derivative-free optimization methods by researchers and practitioners.

Bibliography

[1] Accessed 2022-09. URL: https:www.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html (cit. on p. 24).

[2] Accessed 2022-09. URL: https://docentes.fct.unl.pt/algb/pages/boostdfo (cit. on p. 76).

[3] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky. "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems". In: *J. Glob. Optim.* 31 (2005), pp. 635–672 (cit. on pp. 33–35).

[4] I. Andricioaei and J. E. Straub. "Global optimization using bad derivatives: derivative-free method for molecular energy minimization." In: *J. Comput. Chem.* 19 (1998), pp. 1445–1455 (cit. on p. 1).

[5] C. Audet and J. E. Dennis. "Analysis of generalized pattern searches." In: *SIAM J. Optim.* 13 (2002), pp. 889–903 (cit. on pp. 12–14).

[6] C. Audet and J. E. Dennis. "Mesh adaptive direct search algorithms for constrained optimization." In: *SIAM J. Optim.* 17 (2006), pp. 188–217 (cit. on pp. 12–15).

[7] A. Bemporad. "Global optimization via inverse distance weighting and radial basis functions". In: *Comp. Opt. and Appl.* 77 (2020), pp. 571–595 (cit. on pp. 23, 25, 55, 56).

[8] P. Brachetti, M. F. Ciccoli, G. Pillo, and S. Lucidi. "A new version of the Price's algorithm for global optimization". In: *J. Glob. Optim.* 10 (1997), pp. 165–184 (cit. on pp. 34, 35).

[9] M. D. Buhmann. *Radial Basis Functions: Theory and Implementations.* Cambridge University Press, 2003 (cit. on p. 2).

[10] E. K. Burke and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques.* Springer, 2014 (cit. on p. 24).

[11] F. H. Clarke. *Optimization and Nonsmooth Analysis.* Reissued by SIAM, Philadelphia, 1990. New York: John Wiley & Sons, 1983 (cit. on pp. 14, 15).

[12] A. R. Conn, N. Gould, and P. L. Toint. *Trust region methods*. SIAM, 2000 (cit. on p. 3).

[13] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. SIAM, 2009 (cit. on pp. 1, 5).

[14] T. Cordeiro. "A global optimization algorithm using trust-region methods and clever multistart". NOVA School of Science and Technology, 2020. URL: https://hdl.handle.net/10362/135864 (cit. on p. 25).

[15] A. L. Custódio and J. F. A. Madeira. "GLODS: Global and Local Optimization using Direct Search." In: *J. Glob. Optim.* 62 (2015), pp. 1–28 (cit. on pp. 2, 10, 12, 35).

[16] A. L. Custódio and L. N. Vicente. *SID-PSM: A pattern search method guided by simplex derivatives for use in derivative-free optimization.* 2008. URL: http://www.mat.uc.pt/sid-psm/ (cit. on p. 56).

[17] E. D. Dolan and J. J. Moré. "Benchmarking optimization software with performance profiles". In: *Math. Prog.* 91 (2002), pp. 201–213 (cit. on p. 37).

[18] W. Gao and C. Mi. "Hybrid vehicle design using global optimisation algorithms." In: *Int. J. Electric Hybrid Veh.* 1 (2007), pp. 57–70 (cit. on p. 1).

[19] Y. Hu, H. Qian, and Y. Yu. "Sequential Classification-Based Optimization for Direct Policy Search." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), San Francisco.* 2017, pp. 2029–2035 (cit. on p. 3).

[20] W. Huyer and A. Neumaier. "Global optimization by multilevel coordinate search." In: *J. Glob. Optim.* 14 (1999), pp. 331–355 (cit. on pp. 2, 34, 35, 59).

[21] W. Huyer and A. Neumaier. "SNOBFIT — stable noisy optimization by branch and fit". In: *ACM Trans. Math.* 35 (2008), 9:1–9:25 (cit. on pp. 34, 35).

[22] A. Iske. "Optimal distribution of centers for radial basis function methods". In: (2000) (cit. on p. 23).

[23] J. Jahn. *Introduction to the Theory of Nonlinear Optimization.* Berlin: Springer-Verlag, 1996 (cit. on p. 14).

[24] D. Jones, C. Perttunen, and B. Stuckman. "Lipschitzian optimization without the Lipschitz constant." In: *J. Optim. Theory Appl.* 79 (1993), pp. 157–181 (cit. on pp. 2, 60).

[25] L. Kocis and W. J. Whiten. "Computational investigations of low-discrepancy sequences." In: *ACM Trans. Math. Softw.* 23 (1997), pp. 266–294 (cit. on p. 10).

[26] T. G. Kolda, R. M. Lewis, and V. Torczon. "Optimization by direct search: new perspectives on some classical and modern methods." In: *SIAM Rev.* 45 (2003), pp. 385–482 (cit. on pp. 6, 12, 13).

[27] E. L. Lawler and D. E. Wood. "Branch-and-bound methods: A survey". In: *Operations research* 14 (1966), pp. 699–719 (cit. on p. 2).

[28] M. Leonetti, P. Kormushev, and S. Sagratella. "Combining local and global direct derivative-free optimization for reinforcement learning." In: *Cybern. Inf. Technol.* 12 (2012), pp. 53–65 (cit. on p. 1).

[29] Y. Liu, Y. Hu, H. Qian, Y. Yu, and C. Qian. *ZOOpt: Toolbox for Derivative-Free Optimization.* 2018. URL: https://arxiv.org/abs/1801.00329 (cit. on pp. 3, 59, 61).

[30] S. Marchi and E. Perracchione. *Lectures on radial basis functions.* Preprint, 2018 (cit. on p. 18).

[31] J. Müller. *MATSuMoTo: The MATLAB Surrogate Model Toolbox For Computationally Expensive Black-Box Global Optimization Problems.* 2014. URL: https://arxiv.org/abs/1404.4261 (cit. on pp. 2, 23, 30, 59, 60).

[32] M. D. McKay, R. J. Beckman, and W. J. Conover. "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code." In: *Technometrics* 21 (1979), pp. 239–245 (cit. on pp. 8, 10).

[33] J. J. Moré and S. M. Wild. "Benchmarking derivative-free optimization algorithms". In: *SIAM J. Optim.* 20 (2009), pp. 172–191 (cit. on p. 36).

[34] R. C. Morgans, C. Howard, A. C. Zander, C. H. Hansen, and D. Murphy. "Derivative free optimization in engineering and acoustics." In: *14th International Congress on Sound & Vibration* (2007), pp. 1–8 (cit. on p. 1).

[35] H. Niederreiter. "Quasi-Monte Carlo methods and pseudo-random numbers." In: *Bull. Amer. Math. Soc.* 84 (1978), pp. 957–1041 (cit. on p. 10).

[36] J. Nocedal and S. J. Wright. *Numerical optimization.* Springer, 1999 (cit. on pp. 1, 15, 25).

[37] U. M. G. Palomares. "Searching for multiple minima of bound constrained optimization problems using derivative free optimization techniques". In: *Proceedings of the Eleventh International Conference on Computational Structures Technology.* 2012 (cit. on p. 34).

[38] D. Peri, G. Fasano, D. Dessi, and E. F. Campana. "Global optimization algorithms in multidisciplinary design optimization." In: *2nd AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* (2008), pp. 1–12 (cit. on p. 1).

[39] R. G. Regis. "On the properties of positive spanning sets and positive bases". In: *J. Opt. Eng.* 17 (2016), pp. 229–262 (cit. on p. 8).

[40] S. Rippa. "An algorithm for selecting a good value for the parameter c in radial basis function interpolation". In: *Adv. Comput. Math.* 11 (1999), pp. 193–210 (cit. on p. 21).

[41] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments.* New York: Springer, 2003 (cit. on pp. 8, 10).

[42] R. Schaback. "Error estimates and condition numbers for radial basis function interpolation." In: *Adv. Comput. Math.* 3 (1995), pp. 251–264 (cit. on p. 21).

[43] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104 (2016), pp. 148–175 (cit. on p. 3).

[44] R. Storn and K. Price. "Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces". In: *J. Glob. Optim.* 11 (1997), pp. 341–359 (cit. on pp. 34, 35).

[45] L. Stripinis and R. Paulavičius. *DIRECTGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization.* 2022. URL: https://arxiv.org/abs/2107.02205 (cit. on pp. 59, 60).

[46] L. Stripinis, R. Paulavičius, and J. Žilinskas. "Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT". In: *Optimization Letters* 12 (2018), pp. 1699–1712 (cit. on p. 60).

[47] V. Torczon. "On the convergence of pattern search algorithms." In: *SIAM J. Optim.* 7 (1997), pp. 1–25 (cit. on p. 13).

[48] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Martí. "Scatter search and local NLP solvers: A multistart framework for global optimization". In: *INFORMS J. on Comp.* 19 (2007), pp. 328–340 (cit. on p. 24).

[49] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004 (cit. on p. 2).

[50] S. M. Wild, R. G. Regis, and C. A. Shoemaker. "ORBIT: Optimization by radial basis function interpolation in trust-regions." In: *SIAM J. Sci. Comput.* 30 (2008), pp. 3197–3219 (cit. on pp. 2, 23, 24, 30).

[51] Y. Yu, H. Qian, and Y.-Q. Hu. "Derivative-free optimization via classification". In: *Thirtieth AAAI Conference on Artificial Intelligence.* 2016 (cit. on p. 3).